Stock Management for AAU Bars

AALBORG UNIVERSITY
STUDENT REPORT

P4 Project Report
Group BI406F14
Informatics &
Information Technology
Hand-in 28th May 2014

This page is blank on purpose!

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg East
http://cs.aau.dk

Title: Stock Management for AAU Bars

Theme: P4

Project period: BAIT/INF4 - F14

Project group: BI406F14

Group members:

Mikkel B. Christensen
Morten H. Larsen
Jesper Ingvardtsen
Adam Hviid-Pilgaard
Rasmus K. Schmidt
Roald Jensen
Thomas Guldborg

Supervisor: Saulius Samulevicius

Synopsis:

The purpose of this project was to develop a database system for a specific application. Furthermore experience of software development approaches throughout the entire project had been achieved.
The purpose of our database system is aiding the AAU bars in their work procedure, including counting, updating and viewing the statistics of their stock. To aid the AAU bars in their work procedure empirical data has been gathered and analyzed.
To create our database we used MySQL and an ER diagram to design the database. In addition, normalization of the tables & joining the tables have been applied to respectively creating a simpler database and connecting the tables with each other.
Before the programming phase Lo-Fi prototypes and a navigation diagram have been conducted to get an understanding of the looks and structure of the system
The project is concluded with a website coded in PHP and JavaScript and designed by using bootstrap and HTML.

Circulation: 2
Pages: 77 standard pages
Appendix: 3
Hand in 28/05/2014

The content of the report is freely available, but publication (with source reference) may only take place in agreement with the authors.

This page is blank on purpose!

Preface

The group behind this semester project did it on the 4th semester of Informatics and Information Technology in collaboration with Studentersamfundet and the volunteers from the Friday Bars in the period February 3rd to May 28th, 2014.

The reason for the research project was to shed light upon an on-going internal issue the Friday Bars were having. The administration behind the Friday bars did not use IT. We therefore intend to gain an understanding of their organization in order to create a stock management system to optimize their work.

We would like to thank our supervisor for aiding us throughout this project. Furthermore, we are also thankful for the people participating in the interviews, which made it possible for us to get an insight into the inner workings of the Friday bars.

Signatures

Mikkel B. Christensen	Rasmus K. Schmidt	Morten H. Larsen
Thomas Guldborg	Roald Jensen	Adam Hviid-Pilgaard
	Jesper Ingvardtsen	

This page is blank on purpose!

Reading Guide

This report is divided into 7 chapters. One chapter is dedicated only to theory behind this project and the other chapters describe how the theory has been applied.

The reference method used in this report is the Chicago Referencing Style.

A CD with the source code with documentation is included. The website showcasing our system can be found at http://bi406f14.zapto.org/, and the visitors of the page can log in as either administrator or bartender with the following information:

Administrator login
Email address: admin@aaubars.dk
Password: 1234

Bartender login
Email address: bartender@aaubars.dk
Password: 1234

For more information on installing the system locally, we refer to the readme.txt file on the CD in appendix A

This page is blank on purpose!

Contents

List of Figures **xi**

1 Introduction **1**
1.1 Rich Picture Of The Situation Of The Friday Bars 1
1.2 Existing Systems On The Market 4
1.3 Problem Statement 5
1.4 Problem Delimitation 5

2 Theory **6**
2.1 Burn Down Chart 6
2.2 Interview 7
2.3 Rich Picture 8
2.4 Agile Versus Traditional 8
2.5 Scrum 9
2.6 Extreme Programming 14
2.7 Requirements 20
2.8 ER Diagram 22
2.9 Navigation Diagram 24
2.10 Lo-fi prototypes 25
2.11 Implementation 26

3 Empirical Data **30**
3.1 Requirements Elicitation 32
3.2 Prioritization of Requirements 34

4 Design **36**
4.1 Feature List 36
4.2 Architecture 37
4.3 Graphical User Interface (GUI) 38
4.4 ER-Diagram 41
4.5 Design Conclusion 42

5 Implementation **44**
5.1 Programming Approach 44
5.2 Walkthrough of The System 44
5.3 Technologies Walkthrough 48
5.4 Bootstrap Implementation 50
5.5 Database Implementation 51
5.6 PHP Implementation 56
5.7 Configuration Management 61

6 Project Management **62**
6.1 Introduction To The Chosen Development Method 62
6.2 Our Use of XP and Scrum 62
6.3 First Sprint 63
6.4 Second sprint 69

7 Conclusion **76**
7.1 Discussion 76
7.2 Future work 77
7.3 Conclusion 78

Bibliography **79**

A Digital Copy of Project

B System Prototype Design

C Feature List for the System

List of Figures

1.1 Figure illustrating the rich picture that represents the situation of the Friday Bars . . . 2

2.1 An example of a burn down chart from our second sprint. . . . 6
2.2 A model showing the number of weeks an iteration in XP takes [Larman, 2004,p 138] . . . 17
2.3 A figure showing a typical lifecycle of a project using XP[Larman, 2004,p 124] . 19
2.4 A figure showing the different roles in XP [Larman, 2004,p 124] . . . 20
2.5 Model of an entity and the associated attributes . . . 23
2.6 Model of two entities with a joining relationship . . . 24
2.7 Notation in a navigation diagram . . . 25
2.8 The five basic activities within SCM [Koskela, 2003,p. 10]. . . . 26
2.9 The six parts, which according to Pressman [1997] if used correctly will result in a software project with a high quality. . . . 27
2.10 A very basic example of different branches in a version control [Koskela, 2003]. 28

4.1 Navigation diagram of our system . . . 38
4.2 Prototype of our stock front page . . . 40
4.3 Our prototype of the menu bar . . . 41
4.4 ER-diagram of the system . . . 43

5.1 Picture of the page where it is possible to login to the system. . . . 44
5.2 Picture of the page where it is possible to read news. . . . 45
5.3 Picture of the page where it is possible to get contact information. . . . 45
5.4 Picture of the page where it is possible to get information about a products' sale history. . . . 46
5.5 Picture of the page where it is possible to get an overview of the different stocks within a bar. . . . 46
5.6 Picture of the page where it is possible to perform a count of the stock content. 47
5.7 Picture of the page where it is possible to register a user. . . . 47
5.8 Picture of the page where it is possible to edit your user information. . . . 47
5.9 Explanation of how the code in Listing 5.20 . . . 60

6.1 Our sprint backlog of the second sprint . . . 65
6.2 Our burn down chart from the first sprint . . . 66
6.3 The top menu bar in the system . . . 66
6.4 The page where you can change the personal information about your account . 67
6.5 Our login section . . . 67
6.6 Our sprint backlog of the second sprint . . . 71

6.7 Burn down chart of our second sprint 71
6.8 A sketch of the statistics page 72
6.9 Picture of the update stock function within the system 72
6.10 Picture of the statistics page of the system 73

B.1 Lo-fi prototype of the create user page 82
B.2 Lo-fi prototype of the edit user page 82
B.3 Lo-fi prototype of the forgotten password page 83
B.4 Lo-fi prototype of the front page seen as admin 83
B.5 Lo-fi prototype of the front page seen as user 83
B.6 Lo-fi prototype of the login page 84
B.7 Lo-fi prototype of the menu seen as user 84
B.8 Lo-fi prototype of the statistic page 84
B.9 Lo-fi prototype of the update stock page 85
B.10 Lo-fi prototype of the user list seen as admin 85

Introduction 1

We know that there are many bars, clubs, pubs and so forth everywhere throughout the world. Just in Aalborg there is a whole street dedicated to the bars where there are 24 bars lying door to door competing for guests every day [Jomfru Ane Gade Portalen, 2014].

But it can be a big problem serving all these guests if the bars do not manage their stock properly. For example if the bars thought they had enough beer in stock but realized there was actually not enough in stock when they were about to open for business. It is important to have a good overview of what is in stock and how much the bars sell every time, in order to know the demand and thus be prepared for opening of the bars so they can keep their customers satisfied. Knowing the demand is important for the bars so they can always keep their stock refilled and prepared for the demand. This is a problem we would like to look further into within this project.

We have chosen to focus on the Friday Bars of AAU to see how they handle their stock; this includes how they keep track of what is in stock as well as how they find out how much they need to buy to refill the stock. We have chosen to investigate these bars because we all participate in the Friday Bars frequently. We know from different persons who work at the bars that there are problems within the bars. We have approached this subject with the assumption that the AAU bars do not have a proper stock management system. We assumed that the bars were using simple tools to keep track of their stock and sales and we therefore assumed they would have problems related with their stock and tracking.

In this paper we approached the alleged problem by first examining the problem domain within the Friday Bars to see how the current situation is like at the Friday Bars. Once we had established an understanding of the problem domain, we looked at current solutions to the problems in order to find inspiration for our own solution.

As we had gained an understanding of the problem domain and had requirements ready for our system, we started looking at what practices and tools that could best help us create a system for the Friday Bars. As we had identified practices, tools and techniques from the development methods that we would utilize with this project, we started creating a system, which will be covered more in depth further within this paper.

1.1 Rich Picture Of The Situation Of The Friday Bars

The first thing we did was to get an understanding of the situation of the Friday Bars of AAU. One of the methods we used to get such an understanding was we created a Rich

Picture.

The following rich picture has been made based on theory from Section 2.3

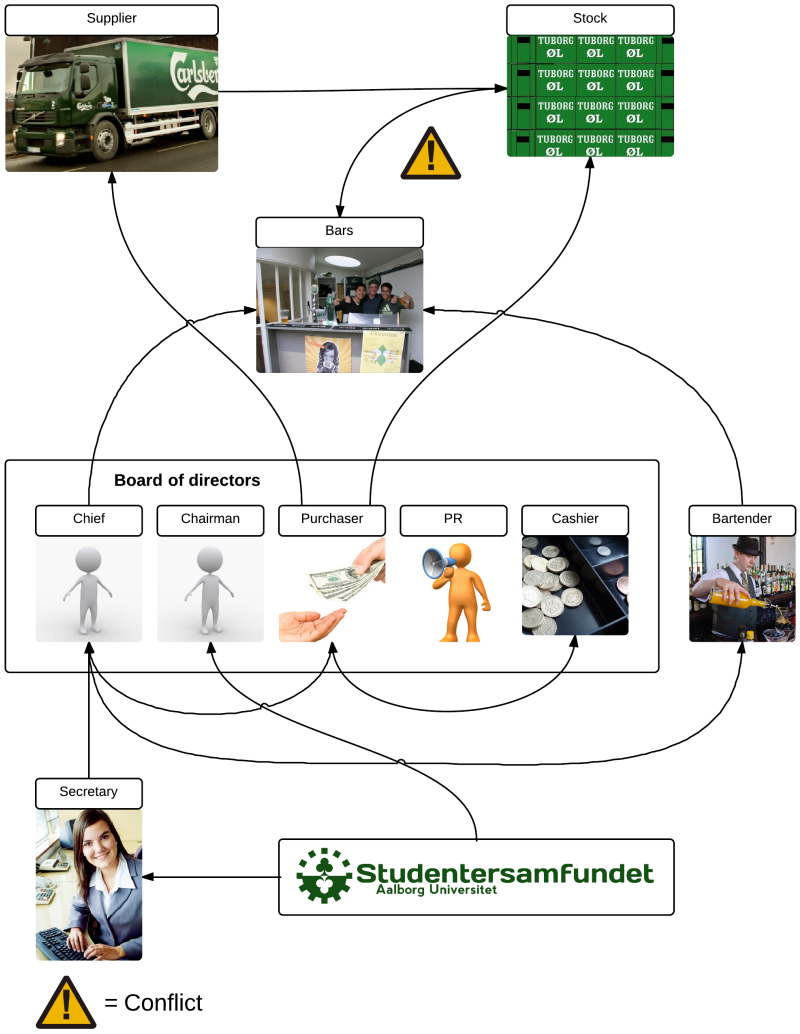

Figure 1.1. Figure illustrating the rich picture that represents the situation of the Friday Bars

This rich picture has been created based on our interpretation and understanding of what the situation is like at the bars, and has since been confirmed and edited according to feedback received from the stakeholders.

As we can see on figure 1.1 we have some different actors in the rich picture.

- Chief
- Chairman
- Purchaser
- Bartender
- PR
- Cashier
- Secretary

- Studentersamfundet
- Stock
- Bar
- Supplier

We can see there is a purchaser in the bars who contacts the supplier, which is the one who delivers all the goods to the stock, and from the stock it goes into the bar where it is sold. There is a double arrow between bar and stock because things that are not sold in the bar can be put back into the stock - While this does not happen all the time at the bars it is still possible.

From the rich picture it can also be seen that there is a chief within the bars. The chief is talking to the purchaser about what needs to be bought to fill up the stock. It is a two-way communication because they are always talking together when they have to figure out what they should get for the opening of the Friday Bars. The chief is also linked with the bar because he is the one who has the overall responsibility when there is an event or if the bar is open. He is also responsible for ensuring the bartenders come to work. The bartenders reply to him every time they can or cannot work at the bar - therefore there are two arrows between these two persons on the rich picture, because it is a two-way communication. At last the chief is responsible for handling the money over to the secretary so the secretary can keep a count of how much the bars are earning.

The chairman has the overall responsibility of everything. He is the head of the bars and is the one who gets in trouble if anything is not running, as it should. He is talking to the board of Studentersamfundet and Studentersamfundet is contacting him whenever they need to get in contact with one of the bars.

The purchaser is responsible for contacting the supplier when they need to buy more items for the stock and the purchaser is also responsible for checking the stock and what needs to be refilled.

The bartender is responsible for sales at the bar.

The Friday Bars also have a PR person. This person is not connected to anything because the PR's function is to get in contact with the customers in order to get them to attend the Friday Bars when they open.

The cashier is making the budget every 6 months so therefore he is in contact with the purchaser to know how much money they have used on buying wares, so he can make a budget for the next period in order for the purchaser to know how much he can spend on wares.

Lastly the secretary is making the overall accountings because this person is authorized from the government to do so. The secretary is in contact with the board of Studentersamfundet because they have to know if something is not right so they can give the message to the chairmen of the bars.

As can be seen from the rich picture there is a conflict between the bars and the stock.

This is because they have separate stocks in the bar as well as a real stock somewhere in a room. This means that there is no tracking on how much there is in the stock in the bar, and how much there is in the real stock. Therefore there is a problem in keeping track of how much stock there is in total which means there is a problem sometimes when they have to order new wares because they do not know how much they have actually have in stock.

1.2 Existing Systems On The Market

With the understanding of the situation achieved through the rich picture we looked into existing systems on the market that could be solutions to the problems and conflicts found from the rich picture of the situation of the AAU bars.

Before developing a new system, it is important to investigate what similar systems are currently available on the market in order to get some inspiration of what features the new system could contain. We have investigated several systems on the market, but instead of describing multiple systems we have chosen only to describe the system AdvancePro.

AdvancePro

AdvancePro offers a wide range of features and is rather comprehensive. In addition AdvancePro also uses their customers to continuously get feedback on their product. By doing this customer involvement AdvancePro is able to adapt their system in relation to the customer needs. The following is based on information from the website www.advanceprotech.com.

AdvancePro is a software solution for inventory management. This software solution integrates the buying, selling, warehousing, and shipping processes in one system and hereby makes the processes easier and more efficient. In AdvancePro you are able to set up unlimited products in the inventory. With this software you are also capable of setting an alert, when the stock of a given product is on a certain level. In continuation hereof you can use an automatic re-ordering feature, which allows re-ordering when the stock inventory reaches this certain level. Furthermore you can update the stock levels by using excel import tools.

Additionally, AdvancePro offers addons like “Reports/Business Intelligence”, “Pricing Management”, “Vendor management”.

The Reports/Business Intelligence addon serves as an efficient reporting system, which delivers real-time reports and intelligence in order to make the right decision. These reports consist of information of purchasing, warehousing, sales, shipping, customer sales and tracking.

The Pricing Management addon facilitates pricing the products. The add on enables assigning the price for the products in proportion to the customer groups, percentage of discount and calendar based sales.

The Vendor Management addon comprises management of vendor relationships in order to help make the right decision and thereby achieve accuracy and efficiency when buying the products. This is done by assigning multiple vendors to the system so that the system can compare the vendors and hereby find the best price for a given product.

With the knowledge of what we knew of existing systems on the market and the fact we understood the situation of the problem domain, it was at this stage possible for us to find inspiration to functions we could use in the system. The elicited functions are covered in section 3.1.

Granted we at this point had an understanding of the problem domain and existing systems that could be a potential solution to the conflicts, we looked into defining a problem statement as guideline on how we could create a satisfactory system for our stakeholders and ensure the requirements of the study regulation were met.

1.3 Problem Statement

With the understanding we have achieved of the Friday Bars and their current situation, we have come up with a problem statement that seeks to solve the problems at the Friday Bars. The problem statement is the following:

How do we facilitate an IT-system that works with a database, that is able to keep track of the bars stock and make a statistic that is able to tell how much the different bars have sold over time?

Within the problem statement we have these points that we would like to answer:

- How can a development method structure the course of our project?
- How can we develop an IT-system with a database that supports the stock management of bars?
- How do we ensure the quality of the system continually?

1.4 Problem Delimitation

As stated previously we have chosen to work with different problems inside the AAU bars. Since having to work with bars in general would be a too wide problem area, we have instead chosen to only work with the different AAU bars within this project. We have chosen Aalborg University's bars because they are representational for the general problem domains within bars. Because of the small amount of time we have, we have also decided to delimitate ourselves to only look at the problems regarding stock and statistic of sold goods. We know that there are more problems within the AAU bars, but we have chosen to only look at the above-mentioned problems.

Theory 2

In this chapter the paper will go in depth with all theory that has been applied throughout the project. The chapter will not focus on how it was applied; the purpose of this chapter is to only outline what theory has been applied to the project.

2.1 Burn Down Chart

This theory written with information from [Mittal, may].

A burn down chart is a graph which depict the iteration's ideal remaining effort versus the actual remaining effort, both measured in days. The actual remaining effort is the sum of remaining time of tasks, which is logged day by day and the ideal remaining effort is the available days left of the iteration. The two graphs together provide a daily status of the progress and hereby shows whether the actual remaining effort is ahead or behind the ideal remaining effort. If the actual remaining effort is behind the ideal remaining effort, then adjustments have to be made e.g. working late, remove some tasks or assigning additional resources.

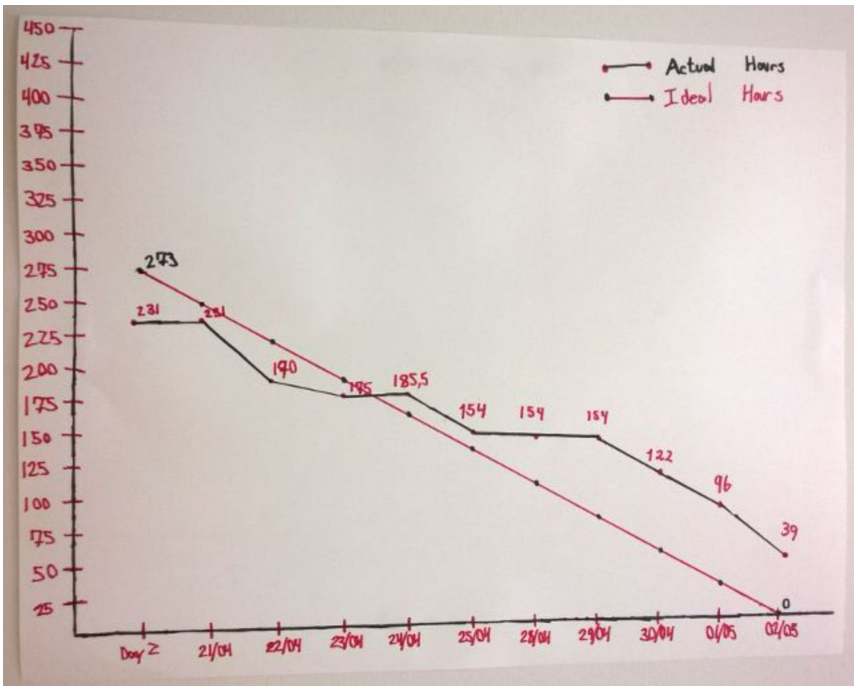

Figure 2.1. An example of a burn down chart from our second sprint.

.

As it appears from the burn down chart above, the graph of ideal remaining effort is blue and the graph of actual remaining effort is red. The x axis is the remaining effort and the y axis is the number of days which have elapsed. At day zero nothing has been completed and remaining effort is hereby the same for both ideal remaining effort and actual remaining effort. At day 20 the iteration ends and by then the remaining effort should be at zero.

2.2 Interview

The focus of this section will be on the theory behind interviews. We look into the advantages and drawbacks of different interview structures.

An efficient way of gathering information about certain needs and experienced problems within an interested party would be to have a simple conversation with them, which is achievable by interviewing them. When doing an interview, there are different ways on how to construct it. It can be a structured interview which uses questions that is prepared in advance and follow this setup without deviation. Meaning there is no flexibility in the structure for the interviewer to deviate from the prepared questions [Benyon, 2010,p. 186].

Another way of structuring an interview is doing it semi-structured. By doing so, the questions are prepared beforehand. This structure introduces a more flexible approach for the interview because the interviewer is able to digress from the prepared questions. This flexibility makes the structure more appealing since it yields a more representative image of the problem area and it is possible to get a more clarified response, if deemed necessary by the interviewer [Kvale, 1996].

Unstructured interview is yet another way of structuring an interview. As the name suggests there is no questions or topics beyond the subject at hand. This way of structuring an interview is good when the need for minimizing assumptions is key, and where no or little background information available [Benyon, 2010,p. 187].

Disadvantages with using interviews as your primary source of data gathering, is that people are limited in what they can remember, they are only able to express certain aspects of the current activities of which they are aware and they may have difficulty describing complex work processes. Yet during an interview you will be able to reach a deeper level of detail on the subject at hand in comparison to other qualitative data gathering methods [Benyon, 2010,p. 207].

We chose to hold semi-structured interviews because of the given situation. We needed to define certain aspects of the problem area and by using a semi-structured approach to the interviews we were able to gather the needed information to define a proper understanding of the problem area while we were also able to elicit requirements out of the interviews.

When doing an interview a good practice is to do an interview guide. An interview guide contains questions for the interviewer to follow during the interview. The importance of the guide changes according to the structure of the interview. The guide can contain

a set of rough topics, or it can be a structure of carefully formulated questions. When conducting a semi-structured interview guide, questions become loose and suggestive for the interviewer.

> *"It will depend on the particular design chosen whether the questions and their sequence are strictly predetermined and binding on the interviewers, or whether it is up to an interviewer's judgment and tact how closely to follow the guide and how strongly to pursue an individual subject's answers."*
>
> - [Kvale, 1996,s. 129]

2.3 Rich Picture

A rich picture is a model to visualize a messy or complex situation by using icons and images for the different aspects of the situation on the basis of gathered information about the organisation. The rich picture outlines the information flows, communication and human activity without a profound structure or formal notations of the relationships [Berg and Pooley, 2013,p. 363-364] [Mathiassen et al., 2001,p. 25].

Information flows communication and human activity are able to cover parts of the organisation, processes and conflicts within the organisation which can later be used in the information system [Berg and Pooley, 2013,p. 364].

By using a rich picture the understanding of a situation across culture, education and culture will be enhanced, because people easier perceive symbols and drawings rather than words [Berg and Pooley, 2013,p. 363].

2.4 Agile Versus Traditional

When creating software there are two ways that the development team can approach the development with: going agile or traditional. To be able to understand the difference between the two development approaches a direct comparison is needed. However, it is important to ensure that we understand what a development approach is before we start comparing the two approaches in software development.

As mentioned earlier there are two types of software development approaches: the traditional development method which focuses on analysis, documentation, techniques, and design. This focus tends to make traditional approaches use a big part of the development process to be solely on documentation whereas the agile approach is more focused on short meetings, programming and iterative driven development. The focus of agile development tends to be more flexible when it comes to changes within the problem domain and has little to no documentation.

The approaches promise two very different things. According to Barry Boehm from University of Southern California, the agile development methodologies promise a lower

possibility of defect in the software, less wasted time in the development and the possibility of requirements being flexible to changes. The traditional development methodologies is more plan-based and therefore promise more predictability, stability, and high assurance. None of these methodologies are without shortcomings which must be looked into to avoid project failure. Agile is made to be able to deal with change and is built around shared vision of the objectives of the project between the members of the development team, so that everyone involved in the development of the software have complete understanding of what should be achieved and how. But agile has a hard time dealing with complexity and to some extent conformity. It does not fit very well into larger more complex projects. The traditional way of software development handles conformity by investing in extensive documentation, but it has large problems with changeability and the increasing complexity in systems.

The following sections describe different methods that can be used when working on a project, and why they are suitable methods in different situations. [Boehm and Turner, 2003] [Persson, uary]

2.5 Scrum

Scrum is an iterative and agile software development method. It promotes strong self-directed teams, daily team measurement and seeks to avoid a direct following of other processes. Scrum is a method that emphasizes a set of practices and project management values, thus making the method an easy complimentary method along with other methods like – as we are using in our project - Extreme Programming. Some of the key practices that Scrum include are:

- Self-directed along with self-organizing teams
- Daily stand-up meeting for the team to get an understanding of the progress of the sprint
- Sprints last around 30 days
- At the end of each sprint the stakeholders should see what has been worked on during the sprint
- Each sprint is determined through adaptive planning

[Larman, 2004,p 109-111]

The Scrum life cycle consists of four phases: planning, staging, development and release. During the planning phase, the team establishes the vision for the product, they set up their expectations and funding is secured. Furthermore, a Product Backlog is initiated. During the staging life cycle the team identifies more requirements and start prioritizing them for the first sprint. In the development life cycle a system is implemented after each sprint. The team also holds the daily Scrum meetings, define the Sprint Backlog, and add tasks or correct estimates of time spent on each task during the sprint. Lastly, during the release life cycle the team ensures the product is operational. Here the activities of the teams are documented and marketing and sales of the product is worked upon [Larman, 2004,p 113].

Product Backlog

A Product Backlog is an ordered list of requirements needed in a product. The Product Backlog is also a source of requirements for all changes to be made to a product. It is in the product owner's responsibility to maintain the Product Backlog; this means which content to include, availability of the product backlog to the team (so they can always see it) and the ordering of the requirements in the Product Backlog. We will describe all roles in Scrum later. A Product Backlog is evolving and ongoing changes can be made. The Product Backlog lists all functions, features, fixes, enhancements and requirements that make up the changes to be made to the product [Schwaber and Sutherland, 2012,p 12].

Working with the Product Backlog in the Team

The team can set up a task board in order to have a combined visual representation of the Product Backlog and Sprint Backlog. The team are the only ones who maintain the task board, as it is them who work and update the task board constantly.

The task board contains four columns: Stories (all the elements from the Product Backlog), tasks to do, work in progress and lastly 'done' stories. The first column contains all the stories the team and the Product Owner want done in the sprint in prioritized order. The second column contains all the tasks within a story that the team needs done before a story can be completed.

The third column contains all the elements that are work progress. When a team member starts a task, he moves the task from the second column to the third column to indicate that the specific task is now worked upon. Tasks that are not complete on the first day are moved to the work in progress get a red dot, so nobody will wonder why it is still on the board. The red dot also helps the Scrum Master in locating blocks in the team so he can do something about it. In the last column, all the complete tasks are put. The team should define the definition of a complete task [Glogler, 2011,p 11-12]. To ensure transparency, all members must have a shared understanding of what it means for work to be done. It is expected that the Scrum Team's definition of done includes more stringent criteria to ensure higher quality [Schwaber and Sutherland, 2012,p 15].

Sprint Backlog

The Sprint Backlog is a set of Product Backlog items selected for delivering the product increment and realizing the goal of the sprint. The Sprint Backlog is a reflection of what functionality the development team want in the next increment and what work that requires.

An increment is the sum of all the Product Backlog items completed during sprints. The increment must be done in a sense that it is in a usable condition [Schwaber and Sutherland, 2012,p 14].

2.5.1 Roles In Scrum

The Scrum Master

In a Scrum Team, there is always a Scrum Master. His objective is to improve the productivity of the team. The Scrum Master works more closely with the product owner in order for both parties to always have an understanding of the progress of the development. Other tasks for the Scrum Master to do, include making sure that the stakeholders understand the agile way of doing all work. Contrary to what one might think of a Scrum Master, a Scrum Master does not take responsibility of the delivery of the product.

Team

In contrast to the Scrum Master and his non-responsibility of delivering a product, the team has such responsibility along with the quality of the product. In collaboration with the end users and the customers, the team also create the Product Backlog. Once the Product Backlog is complete, the team analyses its content in order to gain an understanding throughout the entire team so that the team can make a proper Product Backlog. While the Scrum Master also works with the Product Owner, the team also works together with the Product Owner in order to define the best strategic direction of the product development project.

Product Owner

The Product Owner is the person who communicates the main characteristics and vision of the product. The Product Owner is also the one who accepts the produced content at the end of each sprint cycle. It is the task of the Product Owner to ensure that the team has focus on the most important items of the Product Backlog to ensure optimal development. The Product Owner shares the same goals as the team: to deliver a product of high quality. Therefore, the product owner is expected to work closely together with the team and must henceforth help the team with information promptly if they require it. Seen from the perspective of an organization the product owner's responsibility lies in netting in a return on investment; he is what drives a project from the business point of view.

Customer

The customer is the one the entire Scrum Team aims to deliver a product to. In organizations, customers are typically executive managers who buy software from other software development companies.

End User

End users are a role that several people can cover. The end user could potentially be the real end user; however, it could also be a domain expert or consultant who know what they are talking about. The end user is the person who tells the team what he expects from the product.

Manager

A manager's task is to create the right environment for the Scrum Teams to work in. Furthermore, it is their task to ensure that there is structure and stability within the teams [Glogler, 2011,p 13-14].

The paper will now further describe some of the key practices that make up Scrum.

2.5.2 Scrum Practices

The Sprint

The team defines the length of the sprint. According to Larman he points to sprints lasting, "[...] usually organized in 30-calendar-day iterations" [Larman, 2004,p 117]. However, it can vary depending on the size of the tasks at hand and what the team evaluates it might take.

Planning and Staging

During these life cycles of Scrum, all stakeholders can have a say when creating the Product Backlog. This includes creating a list of features and use cases etc. During these life cycles a Product Owner is also designated the owner of the Product Backlog and prioritizes what must be done for the first sprint. Furthermore, during this life cycle in Scrum there should be generated enough work for the first sprint; and possibly a lot more for the next ones too. As there have been conducted more iterations a Release Backlog is also created. The Release Backlog is a subset of the Product Backlog, which makes up the Product Release.

Sprint Planning

Before a new sprint is planned, meetings are held where firstly all stakeholders meet to prioritize the Product Backlog and Release Backlog before choosing what goals to complete for the next sprint. In another meeting, the Scrum Team along with the Product Owner consider how to best achieve the specifications and then create a new Sprint Backlog of tasks to meet the goals. The Sprint Backlog is subject to be updated if the team discovers new tasks. Usually the backlog is updated during the daily Scrum Meetings.

During sprint planning the team can use Planning Poker for estimation of the time to complete a task. It involves giving cards to the members of the group, and then they present how long they personally think it would take to reach the goal. Those who come with the highest and lowest estimate then reason their choice. After a discussion, the team should come to an agreement of how long they believe the task takes [Persson, 2014].

Self-directed and Self-organizing Teams

As mentioned in the key points of key practices of Scrum there is the self-organizing teams who have control of mostly everything they do. During sprints both management and the Scrum Master cannot guide the team in how they should fulfill the goals of the sprint. During the sprint-phase the Scrum Master can only do tasks such as taking decisions when

the team requests it; otherwise the team is in complete control of their work and the order in which they complete the sprint. The team is granted full authority and resources to solve their own difficulties.

Scrum Meeting

The Scrum Meeting is held same time and place each day. The meeting is held in a circle where each team member answers the same special questions every day. The meeting usually lasts 15 minutes. [Glogler, 2011,p 10]. As mentioned in the key points at the start of this chapter the meeting is a stand-up meeting. The questions answered are:

1. What have you done since the last scrum?
2. What will you do between now and the next scrum?
3. What is getting in the way of meeting the iterations goals?
4. Any tasks to add to the Sprint Backlog?
5. Have you learned or decided anything new, of relevance to some of the team members?

Larman adds the fifth question to provide efficiency to the environment of the team, because the team is just as much working to improve and learn along the way – which, as Larman believes, is vital to agile development. The fourth question also complements the point that a Sprint Backlog is subject to evolve continuously during the sprint [Larman, 2004,p 121].

Sprint Review

At the end of a sprint, there is a review meeting where the team, Scrum Master, Product Owner and other stakeholders attend the meeting. At the meeting, a demo of the product is presented. One of the purposes at the meeting is to inform the stakeholders of the system functions, design, strengths, weaknesses, effort of the team and what problems the team believe they might have in the future. The meeting encourages feedback and brainstorming, however the entire Sprint Review meeting should take no longer than four hours [Larman, 2004,p 120].

Sprint Retrospective

Sprint Retrospective is a meeting held for the team in order to create a plan for improvements on how to work in the next sprint. The Sprint Retrospective meeting is held after the Sprint Review. The purpose of the meeting is to find out how the sprint went in regard to people, relationships, tools and their processes. The meeting also identifies what went well and aims to create a plan on this for a potentially improved sprint in the next round [Schwaber and Sutherland, 2012,p 11-12].

2.5.3 Scrum Values

Openness

Because of the daily meetings, external people can attend and observe the progress and issues that the teams are currently having with producing the product. During the scrum

meeting, only the scrum team can talk whereas anyone else must remain silent [Larman, 2004,p 119].

Responsibility & Commitment

Because of the daily meetings the person who states what they are working on and intend to work on for the day, will feel they have made a social promise to the team and as such seek to aim to follow through with the task. Therefore, the team, given full authority, are committed to their defined goal for each sprint. Management and the Scrum Master should not introduce more work to the working Scrum Team, but must provide the teams with all the resources that they require.

Adaptivity & Focus

Based on the philosophy that software development is a creative and unpredictable way of developing, Scrum is facilitated to adapt to change. In this way a development team is able to measure and respond fast if necessary.

Risk Prevention

The daily meetings are helpful in keeping up with what elements are blocking tasks for the Scrum Teams. As such, the team can quickly respond and solve the blocks in order to optimize work.

Learning Environment

Based on the questions that all team members must answer at each meeting, people are able to continuously improve and learn because all tacit knowledge becomes spoken and shared between the whole team.

2.6 Extreme Programming

While Scrum acts as one method of software development we also looked into more development methods that we could use or find practices from. This led us to researching Extreme Programming.

Extreme Programming is an agile development method which is used to create software quickly and is a suitable method for team collaboration. XP is a method that focuses on customer satisfaction through many quick creations of high-value software, sustainable software development techniques and it is flexible to quick changes in the software. It is usually used on small team projects and it aims at delivering a product in under a year. Many programmers use this method because of the many programming techniques that are within the XP method. Programmers feel safer and more confident when using this method because it supports fast change in the software, so even late in a project it is possible to change code quickly without it adding too much extra work. XP aims at delivering software fast, which has a high business value as well.

Extreme Programming is created upon four values which are: communication, simplicity, feedback and courage [Larman, 2004,p 137].

Communication

Communication within XP aims at improving the development team's communication. There are different tools for this purpose. XP also concerns collaboration with the customer. To optimize communication there are the following tools:

Pair Programming

Pair programming is having two people writing on code together in real time. Pair programming is helpful because you always have two programmers sitting together. One is observing while the other is writing code in real time. They then switch from time to time, so both developers get to overlook each other. This helps because they can reduce errors in the code and thereby reduce time to release.

Daily Stand-up Meeting

The daily stand-up meeting is similar to the daily Scrum meeting and is used to communicate within the team. It is used to get a status on how much people have done and what people needs to do, or intend to do for the day. It is at this meeting that people can talk to each other and tell if they are having problems with their tasks, or if there is anything in the way for them to complete the tasks.

The Planning Game

The biggest planning process in XP is called the planning game. The planning game occurs before every iteration begins. This often happens every once a week and it is here you are planning which requirements should be in the next delivery. This is decided by both the managers and the customer. There are two different types of planning game; the release planning game and the iteration planning game. The release planning game is done to find what should be in the next release to make the best value software. This is done by having the customer writing story cards and describe what they would like in the system, and having the developers estimate upon it. There could also be story cards from before, the customer then has to choose what to have in the next release, either by setting a date and then adding cards that match the estimated time, or by choosing cards and from the estimated time of the cards then calculating the release date from this. The iteration planning game is about choosing which stories should be in the next iteration, planning and allocating tasks. Before every iteration the customer chooses which stories one want to have done, each programmer will then create a tasks list that fulfill the stories the customer has chosen. The programmers are to estimate upon their tasks and if they are rated to more than two days, then it will be refactored to shorten it. Refactoring will be covered later.

Acceptance Tests

Acceptance tests is a way of testing the system. It is automated tests, meaning that it has to test itself so no human inspections are necessary. It has to make a binary output with a pass or fail result. The acceptance tests are written in collaboration with the customer who defines what the code needs to do to pass the tests. This way you always have communication with the customer because they have a finger in making the system as well.

Simplicity

Simplicity is best when using XP, not only keeping the software simple but also the management tools and requirements as well. You can do different things to keep the system simple:

Documentation and On-site Customer

XP does not require much documentation, meaning that you do not need to make a lot of work before beginning the coding process. Instead of doing a lot of information gathering at the beginning of the project, it is better to do an oral communication, eliciting requirements for the system. If documentation is needed it is in form of what needs to be in the next iteration. To gather requirements you can also choose to have an on-site customer, who is viewing the coding process so that the system becomes exactly as the customer wants and you are able to rapidly make new software. XP does not dictate that you should have no documentation at all, but you should use on-site customers to compensate if there is a lack of documentation.

2.6.1 Feedback

To avoid making the wrong system or having much bad code, you want feedback to make sure that you are on the right track. You want continuous feedback from the customer so that you are always making the system to the customer and not forgetting them in the process. To get feedback for the system, these tools can be applied:

Unit Testing and Test Driven Development

Test-driven development is about making tests that the code has to pass before the code is written. It is not a process where you write code and test it, but rather making a test and then write the code for it, making sure that the code is always working thereby making the programmer more secure in writing the code.

Unit testing is used to test the individual units of the code, making sure that every bit of the code is passing a test before it becomes part of a larger system. This means that you want to test if what you think the code will do, is actually what the code does. In this way you are avoiding errors in the system and the unit testing is giving you feedback on the code.

Story Cards

A good way to elicit requirements and managing the project is to use story cards. Story cards are paper index cards where you can write short questions upon, and the customer is then able to write short answers to this question by which you always have the customers within the project. It is a way of communicating with the customer. It also keeps you on track with the system, because you have some requirements that are needed in the system or something that the customer wants to have. It is important to note that story cards is not use cases, which often shows features in the system. Story cards are simply something that the customer would like to have.

In XP, oral communication is preferred and that is where the story cards come into the picture. It is simply a promise to talk to the customer who wrote the card, by the developers implementing it. People have different opinions about how much effort a story should take. Some say that the effort taking to do the story can be from between two days up till two weeks, others say that stories should be estimated in units of one, two or three weeks.

Short Iterations

We can see from figure 2.2, that the recommended length of the iterations should be between 1 and 3 weeks.

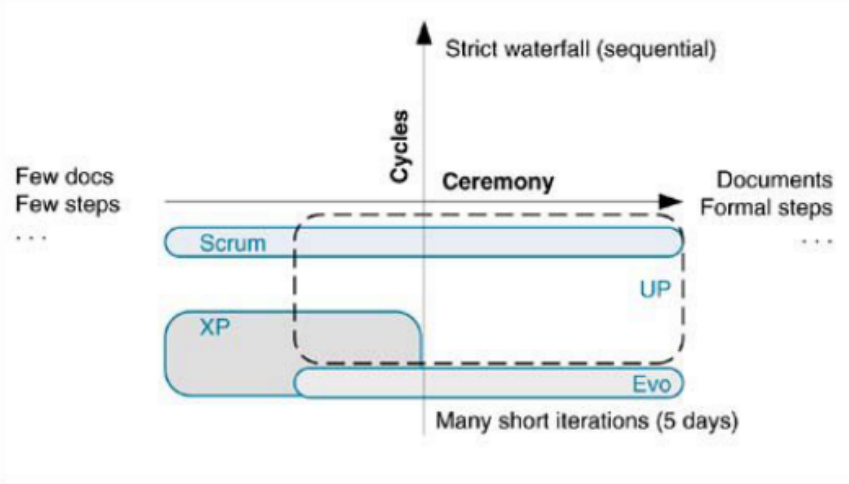

Figure 2.2. A model showing the number of weeks an iteration in XP takes [Larman, 2004,p 138]

XP has shorter iterations than for example UP or Scrum, which has slightly longer iterations. We can also see from the model that XP has few formal steps to go through, where Scrum or UP has many formal steps or a documented guide to go after step by step. You are using the short iterations because you want to implement code after every iterations in the system, but also because having the short iterations makes you able to do quick changes in the system and you can check for every iteration that the customers needs are in focus.

2.6.2 Courage

You need to have the courage to develop fast and make fast changes when necessary. Having many unit tests, acceptance tests and continuous integration is part of enabling the possibility of having rapid and radical changes in the software. Without these it will take a very good programmer to go deep into the code to change it, without having any complications is very difficult.

Continuous Integration

Continuous integration makes sure that every tested code becomes part of the system, so you never miss any code somewhere in the process. You implement code several times a day into the system because you want to prevent implementation problems late in the process. When implementing code several times a day you can quickly spot if there is any error in the code that needs to be corrected.

Refactoring

Refactoring often is part of making the system as simple as possible, meaning that large designs with heavy code can be stripped down to the most basic. This means that you write long confusing code into something much more structured and simple. With refactoring you try to avoid repeated code in the system to make it more uncluttered. This way, it is easier for the developers to correct or change the code [Larman, 2004,p 138-154].

These four values are the key points to extreme programming. To sum up, these methods are recommended when using extreme programming:

1. Planning Game
2. Small, frequent releases
3. System metaphors
4. Simple design
5. Testing
6. Frequent refactoring
7. Pair programming
8. Team code ownership
9. Continuous integration
10. Sustainable pace
11. Whole team together
12. Coding standards

[Larman, 2004,p 137]

Coding standards and metaphors.

2.6.3 Lifecycle of Extreme Programming

.

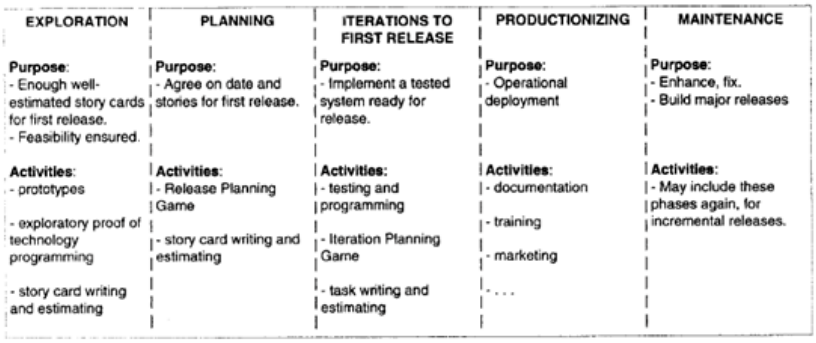

Figure 2.3. A figure showing a typical lifecycle of a project using XP[Larman, 2004,p 124]

In the beginning we have some exploration in form of some story card being written and getting some information from the customer so that you have enough information to begin the project. Then there is some planning in form of release planning game, where you have to choose which stories you want to have in the system and to find out when the project has to be finished. Then we have the iteration which is making use of the iteration planning game, where the customer is choosing which stories to include in the iteration and the programmers are estimating upon this to find out if it can be done in this iteration. The story cards are made in collaboration with the customer to see if the tests and the requirements are as they should be. At last we have the maintenance part. If you are not finished at this state you will return to step 3 in figure 2.3 to start over with another iteration and so forth until the project is finished and the developers only need to maintain and update the system.

2.6.4 Roles in Extreme Programming

There are different roles when using the XP method. We have four different groups with roles as seen below [Larman, 2004,p 145].

Customer

The customer is the one you are making the product for and with. The customer writes stories and also has to accept the tests of the system. Beside this the customer pick stories for release and for iteration. The customer must be available at all time, if the programmer needs to come in contact with the customer.

Development

In development we have the programmer and the tester. The programmer writes the code, the design and the tests. The programmer also does the refactoring and identifies tasks and estimate how long the different tasks takes. Every programmer has to sign up for what they want to do. In XP you are not given a specific task that you have to do, but you choose yourself instead. The tester helps the customer write and develop test to make sure that the customer gets exactly what he or she wants.

Figure 2.4. A figure showing the different roles in XP [Larman, 2004,p 124]

Management

The management consists of a coach and a tracker. The coach has to customize the process, intervene and teach when it is necessary. The tracker is collecting metrics, telling progress to the customer so the customer knows how long the system has come and gives feedback on poor estimates.

Other

In the "other" category, there is only the consultant. His job is to give technical consulting and coach the project if this is needed.

2.7 Requirements

This chapter is based on theory from [Benyon, 2010].

2.7.1 Understanding Requirements

Requirements dictate what a product should be able to do, or what quality the product should have. When defining the requirement specification there is a debate, on which terms used to define the requirements activity. These terms are the following:

- *Requirements gathering* means that requirements lie everywhere, easily found. These are requirements, that with little to no interaction, are to find between the designers and stakeholders needs.
- *Requirements generation* is a more creative activity that de-emphasizes current practices in the problem domain. Here the designers are open-minded when generating new ideas – they do not consider current practices.
- *Requirements elicitation* means that there is a lot of interaction between stakeholders and designers. They work together in order to elicit or come up with new requirements for a product.
- *Requirements engineering* is an approach most often used in software engineering. This means identifying stakeholders and their needs, and then documenting the requirements into something analyzable.

Requirements are divided into two categories: functional and non-functional.

Functional Requirements These are requirements that dictate what a system must be able to do. For example a functional requirement could be that the users should be able to rate a book in the system. A functional requirement dictates the system depending on the context -*must do <requirement>*.

Non-functional requirements These are requirements that give quality to the system. This can include how the functionality works. These requirements contribute to the perception of the system but also contribute to the selling point of the system and to the use of the product. This means that the requirements cover aspects such as aesthetics and design, usability, security, legal restrictions and so forth. As an example you could say that a requirement for a system/product is that it has to adhere to certain laws in countries; otherwise the client cannot purchase the system because it would be against the law to make use of the system. This is just one example of an instance that could hinder sales of the product. A non-functional requirement is defined as something that dictates the system - *shall be <requirement>*.

2.7.2 Prioritizing requirements

When prioritizing the requirements the developer can use the method of classifying requirements with the MoSCoW rules [Benyon, 2010]. These rules are:

- Must have – Requirements that are needed in order for the system to work. Without these, the system would be useless.
- Should have – Requirements that are essential for the system if the development team has more time, but the system can function without these.

- Could have – Requirements of lesser importance. Able to leave these out of current development if deemed necessary.
- Want to have but will not have this time round – Requirements that can be developed later if there is time.

The requirements this report has elicited and prioritized can be found in chapter 3.1.

2.8 ER Diagram

Database design normally consists of a number of different tables and columns. The design of how these tables are made have as of yet not been too concerning. Therefore, by using design techniques, such as an ER diagram to generate the tables and columns it will make a database administrator able to depict the database and thereby give an overview of how the architecture will be. Entity-Relationship (ER) concept is a technique to structure databases and makes it possible for a database administrator to identify certain elements (entities, attributes, relationships, etc.) for the database easily. After generating a model of the database in the form of an ER diagram, the database administrator can translate the design into relative tables and constraints within the database [Teorey et al., 2008,p 13]. An ER diagram is able to give an overview of the database and the relationships between the individual tables. The depiction of the database is also convenient when presenting the database to the customer because they are able to get an overview of how the database is structured in a less technical way.

An ER diagram has certain elements. One of these elements are entities and can be defined as:

> *An entity is a collection of distinguishable real-world objects with common properties.*
>
> - [Teorey et al., 2008,p 13]

The entity represents a table in a database and is therefore the core of the database.

Another element is an attribute and is defined as:

> *An attribute is a data item that describes a property of an entity or a relationship.*
>
> - [Teorey et al., 2008,p 15]

Attributes are represented by circles in ER diagrams and connects with either entities or relationships. Attributes serve as a property for the entity it connects to, thereby defining values for the entity. Taking this into account, an attribute would be a column inside a table.

Certain values serve a more important role than others do. One of these roles, called an identifier, identifies the individual rows within a table and is known as a primary key in the table. Other attributes behave differently by not identifying but describe the entity. Most attributes are simple, meaning they only handle one value but there are types of attributes that define certain aspects of the attribute but in turn also the entity. These types of attributes are multivalued attributes, composite attributes and derived attributes. Multivalued attributes depicts two lines connecting an entity with an attribute in an ER diagram, and is an attribute that can take on multiple values for an entity. An example of this could be if an entity called "Computer" had an attribute called "Programs", this attribute would be a multivalued attribute because a computer can have many programs installed on it, or it could have a few depending on the user. A composite attribute is a set of other simple attributes that collectively describe the property. An example would be the attribute named "name". It is a composite attribute because it is composed of attributes called "firstname" and "lastname". The "name" attribute receives its value from these attributes and without them "name" would not have a value.

A derived attribute performs a calculation based upon another attribute's value. An example could be the attribute "age". It is generally a bad idea to store a continually changing value. Values have to be consistent. So in order to get the age from someone, you need to get his or her date of birth (DOB) since it is a consistent value. From the DOB it is possible to calculate the age of someone and thereby show it if deemed necessary. The derived attribute uses a dotted line in the ER diagram and it is described what the dotted line means in the ER diagram.

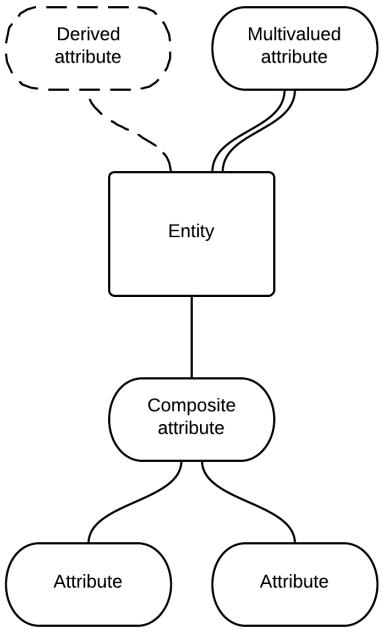

Figure 2.5. Model of an entity and the associated attributes

In figure 2.5 an example of a simple ER diagram is shown. The model shows an entity

with accompanying attributes, where it is possible to see what the individual attributes look like. A more complex diagram would contain several entities and attributes. Entities connect through a relationship, which are represented by the lines between the boxes.

Teorey et al. [2008] talks more specifically, about a binary relationship between the entities. The binary relationship is a simple connection between two entities that define what they have in relation to each other. An example could be that a unique user login to a specific system. A relationship between the user and the system exist through the login sequence. This form of relationship is binary because of the two entities that interact.

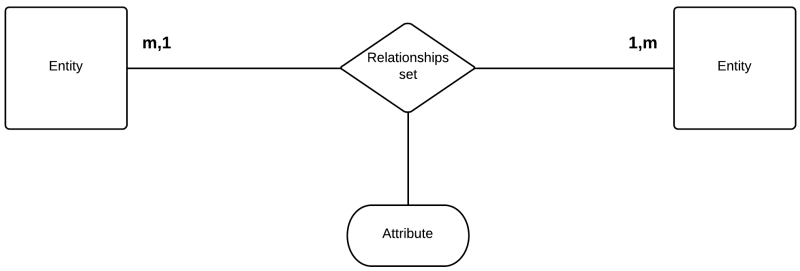

Figure 2.6. Model of two entities with a joining relationship

Figure 2.6 shows a model of a relationship between two entities. Shown in the model there are cardinalities between the two entities. These cardinalities show how to interpret the relationship. There are three types of cardinalities, one-to-one, many-to-one, and many-to-many. The model shows the cardinality, by using either an "m" for many and a "1" for one.

A One-to-one relationship happens when one instance of an entity can reach the connected entity and one instance can reach back. An example could be if a person eats an apple; he or she would be the only one that can eat it and the apple would also only have one person to eat it. A many-to-one relationship happens when several instances of an entity can reach the connected entity but only one instance can reach back. An example of a many-to-one relationship would be an ATM; it has numerous users but is only able to serve on at a time. Finally the many-to-many relationship between entities happens when several instances of an entity can reach the connected entity, and several instances are able to reach back. An example of this would be a website, which can be accessible to many users. And a user can access many websites. Certain relationships sometimes use primary keys from entities to show a connection to another entity. These primary keys are called "foreign keys", because they are primary keys from other entities and serve to identify the connection between the two entities.

2.9 Navigation Diagram

A navigation diagram is a state diagram that shows the dynamic of the user interface as a whole. The diagram shows the various windows and the transitions between them. In a navigation diagram a state is represented by a window. See Figure 2.7.

The state has a name and contains an icon in the form of a picture of the window. On the navigation diagram the transition of state is equivalent to a shift between two windows in the system. The transition of state is represented on the diagram as an arrow, as can be seen on Figure 2.7. On a transition of state the actions on the window are specified. These actions are precisely specified because they are actions that the user must do in order to activate a transition.

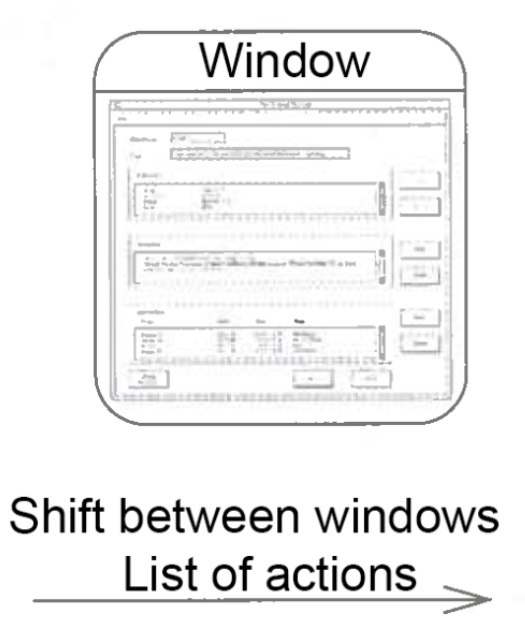

Figure 2.7. Notation in a navigation diagram

2.10 Lo-fi prototypes

A Lo-fi prototype is a simple model that illustrates a system. These are often made on paper and can quickly be made and can represent the most basic design ideas of a system including its structure, key functionality and navigational structure. As the design of the prototype is not fully representational it makes sure that generation of ideas is still possible and new ideas can quickly be incorporated.

Lo-fi prototypes can also be visualized through screenshots with a simple way of interaction, e.g. representation of artificial buttons, which by clicking passes you on to another screenshot. When a design has been chosen it will be used as a guideline for the developing team.

The issues by producing lo-fi prototypes can be the stability. If many people are to interact with it, it may easily fall apart, furthermore it can be difficult to keep the balance between too high levels of details and too low levels of details. Another issue is the flexibility of the Lo-Fi prototypes. If it is not flexible then new design ideas cannot be incorporated. [Benyon, 2010].

2.11 Implementation

2.11.1 Configuration Management

When working with software development one of the big issues can be to work with changes. Depending on the chosen method the different approaches and issues can vary from time to time. When working with the traditional model changes in the software can be rather costly - especially if they are introduced late in the project. If a change is introduced late in a traditional project a lot of existing work is involved and is up for change. This could be some of the documentation not directly involving the source code for the system - as an example:

- Requirements specification
- Design specification
- Test plan
- System documentation
- User manual

After the changes have been implemented in the already existing documents, you would also have to include it into the source code, the testing and the documentation of the code. This is why the traditional method is not very good at adaptation to new requirements.

On the other hand, if you would work with an agile development method you have to be prepared for changes - especially if you do not do much research of the problem domain. One of the key points in agile development is the general willingness to adapt. Often there is a higher need for formal work in a traditional project before the actual coding of the system. Because of this, more work is often done before the customer can start seeing the product - in agile it is different. Here the customer can quickly see some of the product and can give feedback immediately, so that the developers can adapt the system. Apart from a change in the customer's requirements, a change in the platform where the system is running, whether it is a smartphone, a laptop or a third device. Other sources of change could be quality assurance, design improvements etc.

One great way to handle the problems is "Software Configuration Management" (SCM), which according to Koskela [2003,p. 11] consists of the following 4 basic activities found on Figure 2.8.

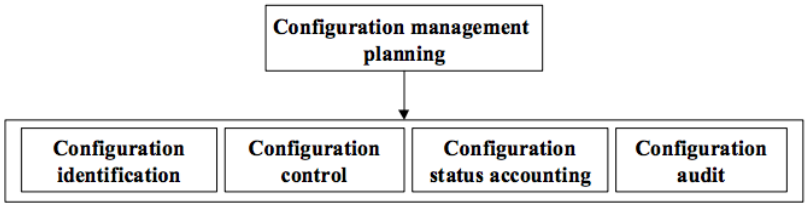

Figure 2.8. The five basic activities within SCM [Koskela, 2003,p. 10].

Configuration Management (CM) was introduced in the U.S. defense industry as a discipline of controlling complex systems. It was used to control the manufacturing

processes - but software projects had also become subject for a rather high complexity which introduced the need for SCM. Although SCM really just is CM for software, there is multiple ways in which the SCM differ from the CM. Firstly software is often easier and faster to change in contrast to hardware products. Second the SCM could be automated taking it to a new level. [Koskela, 2003,chap. 2]

According to Koskela, SCM is one of the key points when it comes to making software projects with high quality, and because of that, it is categorized as a critical element of software engineering. With the systems getting bigger, higher demands for systems and systems becoming more complex, the need for a good SCM strategy is increasing.

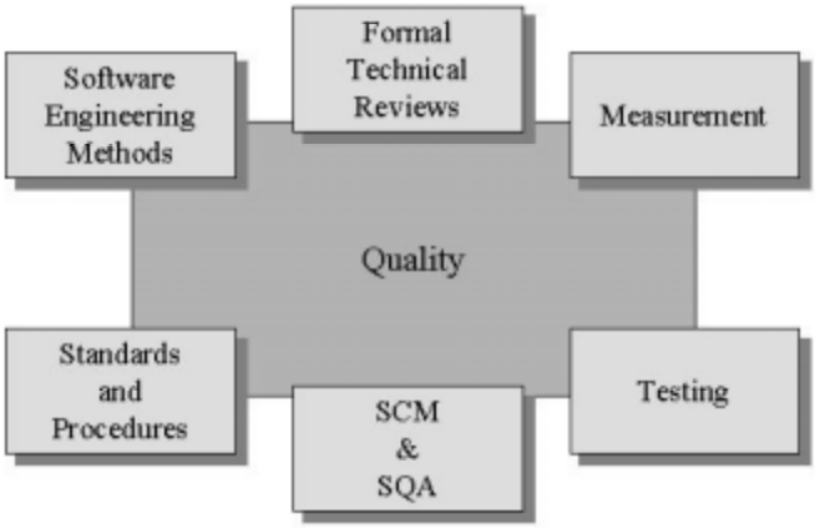

Figure 2.9. The six parts, which according to Pressman [1997] if used correctly will result in a software project with a high quality.

In the following sections the areas which have been used in the project are described.

Version Control

When collaborating on projects you need a sophisticated and well controlled way of handling the different versions of files and source code. By using a version control system one can accomplish precisely this. When you are working with files, which are under version control, all changes will produce a new revision of the file giving a possibility to track and revert changes over time. Besides the tracking of individual files one can create different variants of the software called "branches" and hereby work parallel on systems without interfering with each other.

Depending on the project two different kinds of variants can be introduced, either a permanent or temporary variant. The permanent variant can be made if the developers are going for a version for another platform or operating system, and may work independently after this variation was created. Besides this, you can make a temporary variant which later will be included in the original variant by merging the two branches. This is useful if you have to introduce big changes or simply have to test something that otherwise would

interfere with the remaining developers on the project. An example of different branches can be seen at Figure 2.10 [Koskela, 2003].

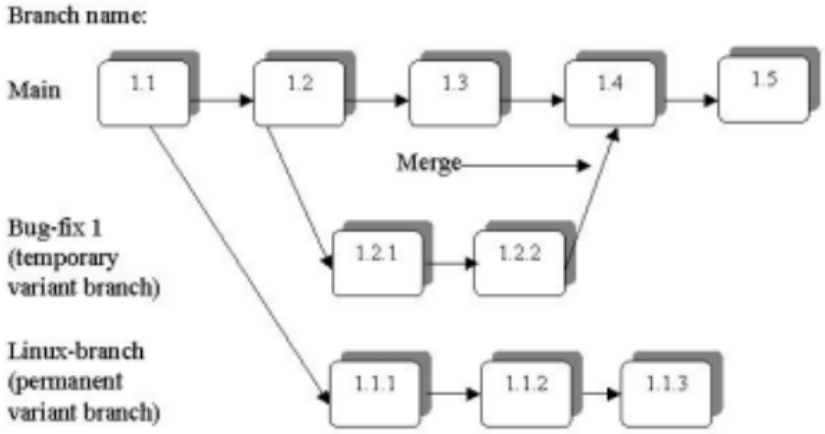

Figure 2.10. A very basic example of different branches in a version control [Koskela, 2003].

Build and Release Management

The purpose of build management is to combine different files into a software build and compile them into a running application. This is useful for different operating systems, platforms or even different versions (alpha, beta, release candidates etc.) of the software. When building your application one can either do it incrementally or from scratch. Incremental builds only compiles the files which have been changed and thereby spares a lot of computing power if it is a large project. When you in contrast build a system from scratch you have to compile the complete system, which may require quite a lot computing power and may take more time. The build management is an area which eventually can be automated with various tools depending on the needs.

When working with software you also need a strategy for handling the different releases of the system and often a release will consist of multiple files beside the build itself.

When working with build management there is a great chance that you would also implement a release management strategy for the project. The different files which have to be included in the release could be configuration files, files for the specific platform or installation programs. Besides this, documentation whether it is electronic or on paper, should be taken care of. Again, depending on the different needs in software projects, tools are available for automating the process. According to Van Der Hoek et al. [1997] the following list of requirements has to be fulfilled from the developers point of view:

- ***Dependencies should be explicit and easily recorded.*** *It should be possible for a developer to document dependencies as part of the release process, even if dependencies cross organizational boundaries. Moreover, this description should be directly usable by the release management tool for other purposes.*

- ***A system should be available through multiple channels.*** *Once a developer releases a system, the release management tool should automatically make it available via such mechanisms as FTP sites and Web pages.*
- ***The release process should involve minimal effort on the part of the developer.*** *For example, when a new version of a system is to be released, the developer should only have to specify what has changed, rather than treating the new version as a completely separate entity.*
- ***The scope of the release should be controllable.*** *A developer should be able to specify to whom the release is visible and to whom it is not.*
- ***A history of retrievals should be kept.*** *This allows developers to track their systems, and to contact users with announcements of new releases, patches, related products, and the like.*

- [Van Der Hoek et al., 1997]

Agile and traditional configuration management

Whether you're using the agile or traditional way of software development you are using more or less the same tools for configuration management but the difference between them is which areas they focus on. Also the people responsible for the configuration management differs from the traditional to the agile development. When working with CM in a traditional way, it is important to make formal documents with details about releases, etc. This is often done by having a person responsible for the different documents and tools for the CM. In contrast to that, according to Schuh [2007] agile CM depend a lot on the individual developer and often include automated tools which will help the developers making builds and releases with continuous integration.

Empirical Data 3

This chapter will show how we came to understand the current problem domain of the AAU Bars, and furthermore the requirements the interviewees had. We gathered the information through interviews with the stakeholders. We conducted the interviews sitting with the interviewees and recorded the sessions on voice recorders. The interviews we conducted are based on the theory as presented in chapter 2.2. The recordings can be found in appendix A.

Roles in the AAU Bars

Through analysis of the interviews we have identified these six roles:

- The Purchaser
- The Chief
- The Chairman
- The Cashier
- The PR
- The Bartender

The Purchaser, is responsible for shopping, so that the stock is adequately filled up for the next opening day. The Purchaser communicates with the chief about which items are needed to fill up the stock. If the chief does not know what is required at the bar, or is unavailable to get in contact with. The Purchaser will have fill the stock with items which he estimates is need. This makes the purchaser the primary contact with the supplier.

The Chief, is responsible for the bar on opening days. This includes keeping track of earnings from the bar, and that the money is handed over to the secretaries. The chief needs to make sure that the bar has enough bartenders as well. The chief is also responsible for the credit card terminal, making sure that everything works. Besides this, the chief also acts as a bartender in the bar and is a member of the board of directors. The Chief is also responsible for planning upcoming events in the bar.

The Chairman, is the highest person in the hierarchy. This person is the one who has the overall responsibility for the bar. The chairman takes contact to the other bars in the AAU bar community, to discuss with other chairmen if there are any interesting events they should create. The Chairman also works as the main contact person to the public. Furthermore the Chairman works as the main formal contact to the secretaries. It is important that the Chairman is able to keep an overview of what is going on at the bar.

The Cashier, is responsible for making the accounting every 6 months, and from this create a budget for the next 6 months, and then deliver the report to the secretaries where it is then approved. As a part of budget planning, it is the Cashier's task to check the stock to make sure that it fits with the anticipated sales. The anticipated sales are based on notes and income from the opening days.

The PR, has the responsibility for informing people about special events and to inform people about the bar in general. Another part of the PR's job, is to take contact to the media and inform of upcoming events. When promoting the bar the PR makes flyers and posters for bigger upcoming events.

The Bartenders, are responsible for sales in the bar. They are "hired" as volunteers each week by the chief to take a shift at the bar.

Communication

There is a great loss in knowledge of knowing how to do things when the old board of directors is replaced with a new one. This can happen for various reasons, sometimes it is because the bar changes all members in the board of directors once a year, and at other times it is because members in the board of directors complete their study and choose to stop being part of the bar. The general communication within the bars is often informal on Facebook or at general meetings. This approach has some problems because the communication tends to drown in funny posts and pictures. This way important information can disappear in clutter, because it is delivered through comments on Facebook. The bars have a private Dropbox for sharing photos from parties, ideas for posters & flyers and guidelines for the bars. These are folders that only members in the board of directors and other people related to the bars have access to. For planning purposes Google Calendar is used to make sure that everyone who is needed at specific events, knows about it.

Stock

Whenever the Purchaser is replaced it is often difficult for the new purchaser to make a good estimation in regards to restocking the bar. However, this is still a problem making correct estimations for an experienced Purchaser. Furthermore, it is difficult for the bars to keep track on their items that are reaching the expiration date, which leads to products going to waste and becomes an extra expense. All members in the board of directors have access to the stock. Because of the lack of a good stocking system to keep track of what is going in, and what is going out of the stock, it is difficult to find out who is to be blamed if some items are missing in the stock. Due to big changes in the number of customers between each week it is even harder to estimate a fixed consumption for the bar. As it is right now, no one is making statistics of the sales, shopping or stock. With the lack of a system to keep optimal track of the stock, there is a potential chance that items will disappear without anyone knowing why. The only system they have for the stock is either pen and paper or Excel on computer, but these systems are only used to keep track of how

much there were on stock before and after the open days in the bars. This information is not saved for later use.

Shopping & Sale

A fixed contract is made with a supplier on an annual basis to avoid having to go hunting after sales each week. This contract only covers beverages such as alcohol and soft drinks. For decorations and similar things such as cleaning articles the bars need to go to other suppliers. There is no contract for these suppliers and this shopping is done in a more improvised manner. As part of the contract with the supplier of beverages, it is agreed that they deliver to the door. It typically takes between two and three business days before an order will arrive. The board of directors decide which things to buy for the individual bar, and as a general rule no new products are bought for testing in the bar. During the opening hours no effort is made to keep track of the sale. After the bar has been closed the responsible will count the stock.

Problems In The Organization

- The lack of a system to check the contents of the stock, makes it hard to ensure that nothing has been stolen, or has simply been counted incorrectly from time to time. This means that items disappear, without any hope of finding out where it has gone.
- The chairman is having a hard time ensuring that the people responsible of the bar actually show up on the opening day.

3.1 Requirements Elicitation

From the interviews with the bars it is now possible to elicit some requirements for our system. These requirements are elicited through an interpretation of the knowledge we have gained of the problem area from the interviews and research of existing systems. The requirements that are found have then been prioritized using the MoSCoW method.

Requirements For The System

We know from the rich picture how much they communicate with each other. From this and the information we have from the interviews that they have lists hanging around with everybody's phone number, we could have a place where a list of contact information is included.

- A list of contact information to everyone associated with the bars

It was stated in one of the interviews that it could sometimes be difficult knowing about events in other bars. The reason they would like to know this is because they would like to know of these events before planning an event for their own bar. From this a requirement to the system could be:

- A way to show if a bar is hosting an event

With our research of existing systems it could be interesting if our system could have the same feature as AdvancePro, where the system tells when a stock is reaching a "critical" amount where it needs restocking.

- The system tells the user when a stock is so low it might need to be restocked

We know from the interviews that there is a great loss of knowing how to do things when they are changing the staff in the bars.

- A way to see what tasks are associated with the different roles

Because communication sometimes seems to drown in the media which they utilize for communication (Facebook).

- A way of communicating with each other through a forum to prevent loss of information

They are using Google Calendar to produce a duty roster.

- The system should be linked with Google Calender for planning

The Bars have some problems with stock items reaching the expiration date, and therefore cannot be sold.

- The system should be able to move expired products into a wasted products category

They are transferring products from bar to bar when one is in need of a product, they then just pay the purchase price to get it. This has to be registered in the system.

- There needs to be a function that updates the system when products are transferred from bar to bar

AAU's bars have a list over products they can buy from the supplier.

- There should be a list over what items are available for the purchaser to order
- The system should be able to update the purchase price from the supplier

They find bartenders who can work through Facebook. All available bartenders tell in their private Facebook group if they are able to work on a specific date.

- The system should show what bartenders are available on a specific date

All the roles in the bars have access to see everything in all the current systems they are using for planning, looking at stock and so forth. However, it is only certain roles that are authorized to take the final decision.

- All roles in the bars need access to the whole system, however only some users should be able to make edits in the system while others cannot.

To make it easier for the bartenders to know the products and their prices, they would like to see the price of the product in the bar.

- The system should be able to show the prices on the products in the stock

The bars are counting the stock before and after opening days to see how much they should have sold every time.

- The system should be able to calculate how many items they have sold
- The system should be able to calculate potential earnings from what has been taken out of the stock
- The system should compare last weeks closing count with the new count to check if there is any loss of products doing the week

The bars would like to have some statistics over the sales of each specific bar to see how much they have sold each week.

- The system should be able to make a statistic of the sales every week
- The system needs to be able to show important events that occur annually to help the purchaser with estimation every year of what needs to be in stock

They have specified a minimum of every product they must have in stock.

- The system should be able to tell how much they need to buy to fill the stock to the minimum

We learned that they do not want to spend a lot of time sitting with their work.

- The system should be simple to operate to limit time spent with the system

3.2 Prioritization of Requirements

We will prioritize these requirements based on what we have time to implement into the system, so the system is satisfactory running. The method we use to prioritize the requirements with is the MoSCoW method [Benyon, 2010]. It is used to divide the requirements into groups depending on their importance. We will then divide the requirements into functional and non-functional requirements as well.

Must Have

Functional

- A list of contact information to everyone associated with the bars
- There should be a list over what items are available for the purchaser to order
- The system should be able to show the prices on the products in the stock
- The system should be able to calculate how many items they have sold
- The system should compare last weeks closing count with the new count to check if there is any loss of products doing the week
- The system should be able to tell how much they need to buy to fill the stock to the minimum
- Depending on the role of the persons associated with the bars, they all do different tasks. Therefore limit rights to some functions in the system depending on their roles
- Some roles in the bars need access to the whole system

Should Have

Functional

- A way to see what tasks are associated with the different roles
- A way of communicating with each other through a forum to prevent loss of information
- There needs to be a function that updates the system when products are transferred from bar to bar
- The system should be able to calculate potential earnings from what has been taken out of the stock
- The system should be able to make a statistic of the sales every week
- The system tells the user when a stock is so low it might need to be restocked

Could Have

Functional

- The system should be able to move expired products into a wasted products category
- The system should be able to update the purchase price from the supplier
- The system should show what bartenders are available on a specific day
- The system needs to be able to show important events that occur annually to help the purchaser with estimation every year of what needs to be in stock

Want to Have, But Won't Have This Time Round

Functional

- A way to show if a bar is hosting an event
- The system should be linked with Google Calender for planning

Non-Functional

- The system should be simple to operate to limit time spent with the system

These requirements are used when designing the system because the requirements are the needs of the stakeholder and are therefore important to consider when designing the system.

Design 4

In this section we will take a look into the methods used as guidelines for the design of the system. This section will help gain an understanding of how this process was carried out. It is important to make clear that design is a wider spectrum than just the graphical surface. This definition covers design all the way from the database design, to functionality in a function list and onto the architecture of the system, which is processed in the navigation diagram. To the top with graphic design which was decided through the work on Lo-Fi Prototyping.

4.1 Feature List

The feature list contains all functionalities needed for the system to be in working satisfactory. To get an overall understanding of the purpose of the system, we sorted the list into primary, secondary and tertiary functionalities. Because we have delimitated ourselves to only look on stock management and statistics, we have decided only to work on implementing the primary functionalities into the system.

The primary focus of the system was to help the user manage the stock in the bar and be able to get statistics of sold products over time. Under fundamental features, a more accurate description of the functionalities can be found at appendix C. The first of these features is user management. In order to manage the users through the interface, the functionality first had to be there to support that aspect of the system. From this, the functionality to login and out of the system was essential as well since having users would not make much sense without such functionality. This furthermore restricts the number of people using the site, by denying them access without a proper account. Access to the system for new volunteers is made by the administrators to make control of the number of users. In the same manner, only admins can delete existing user from the system. We estimated that by allowing only the administrators to manage users, the number of inactive users would diminish.

The different types of users were created in such a way that it depict the situation of the organization in the bars, and as in the organization, a user needs to be able to have a role in more than one bar. An administrator is able to create, delete and edit roles of any user. If a user has forgotten their password, the system needs a functionality to aid the user recover this. Management of the stock is another fundamental feature described in the feature list. The natural place to start is in the functionality that makes it possible to create, delete and edit stocks. Only admins can use this functionality, otherwise it would

be possible for normal users to create and delete stocks. In addition, a stock will be a part of a bar when created. Adding and removing items within a stock is common procedure and therefore a similar functionality is required in the system. For the user to be able to keep track all the different items in all the stocks, it is necessary to add and remove items as they move through the different stocks. This is where the "Receive new items" functionality comes in. A user would need to send a list of items and quantity to the purchaser, when the purchaser makes an order. The last functionality would be to view other bars stocks.

The section for menu in the feature list describes the content of the menu bar of the whole system. It has the option to change settings, has links to user information, a stock overview and statistics. Administrators are able to add new users and all users are able to log out and edit their own profile from the menu bar. The front page of the system delivers the user an overview of the news from Studentersamfundet and the bar that he or she has selected. Admins have the right to changes the content of the bar news, however only news from the bars where they have the role as administrator.

To help the bars estimate their purchasing needs, the system will have statistics showing sales numbers for the individual items sold over time. Under contact list is an overview of the entire user base that works in whatever bar chosen; here you will see a name, a phone number and what bar the user is associated to.

4.2 Architecture

From the understanding we have gotten from the requirements and the function list, we can make a navigation diagram to get an overview of the navigation in the system and the architecture of the system, as it is seen from the perspective as a user. In our case this is a tool to help the development team to understand how the different pages of the site are connected. We use the knowledge we have from the chapter 2.9 with theory about navigation diagram.

When looking at the navigation diagram 4.1 below, which represents our system's architecture, it can be seen that it starts in the top with the users opening the website. The first site that our users will meet is the log in page where the only two options that are available are "Log in" and "Forgotten password". If the users fail to log in they will be able to choose the forgotten password page where the users can acquire their password by using the e-mail connected to the users. The system will then send an e-mail to the email of the users with their account information. If the users successfully logs in they will be sent to the front page of the system, where they will find the news feed that consists of news from Studentersamfundet and the individual bars. From here administrators are able to edit and delete the content in the news if necessary. All users can change from which bar they want the news. On the top of the page the users will find the next element of the navigation diagram: the top menu. Here users can log out of the system, but if this is not what the users want they can choose to go to the stock page or the list of users in the different bars. From the stock page all users can see the different bars' stocks and

which items the stocks contain. Here administrators will be able to create, edit or delete any stock. If this is done the data will be shown in real time. The administrators can add columns and rows and edit fields. As part of the edit function there is a save and cancel button as well.

The page with a list of users gives the users the opportunity to look at information of other colleagues who work at the bars. From this page administrators can create, edit or delete users in the system. The administrators will be able to save the information if all fields have been filled out, and when creating a user the administrators always have the opportunity to cancel. If the administrators or any of the users want to edit an account, they will get the current information of the chosen user, and all fields can be edited if needed, but it is only the administrators who are able to delete a user.

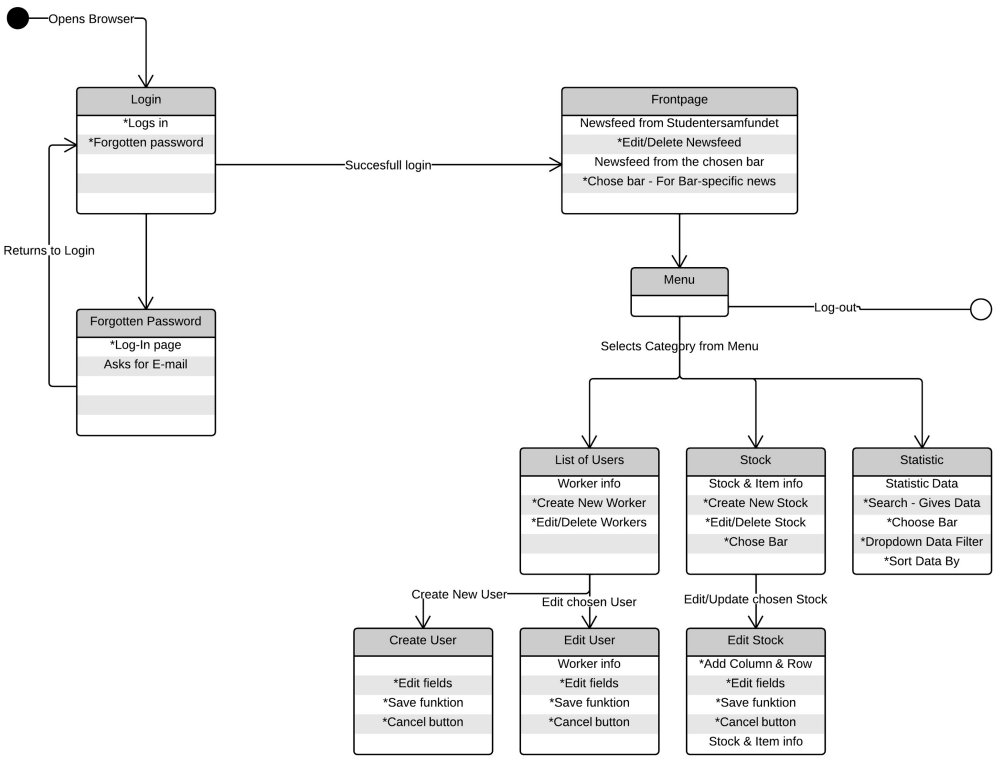

Figure 4.1. Navigation diagram of our system

4.3 Graphical User Interface (GUI)

When designing the graphical user interface of a system it can be helpful to use Lo-Fi prototyping. When not putting too much work into the design of the prototype it is possible to fast and effectively move on to another design if the first was not good enough. When you have chosen the design it is easy to start programming the visual aspects of the system. When you start programming the GUI with sketches of the layout, it is possible to change things if you discover that elements of the original design do not work as intended. The log in screen is, as previously mentioned, the first page that a user will meet when

accessing the web site. On the prototype of the log in screen (See Figure B.6) we see the logo of Studentersamfundet on top of two boxes that the user can fill in with their username and password. Below this there is a button for the user to click on if he or she has forgotten the password of his or hers account. After we started working on programming the interface of the log in screen, it was discovered that a button was needed for the user to be able to log on to the web site. So this button was added during the coding process.

The screen for forgotten password (see Figure B.3) only holds a short text that serves to inform the user on how to get access to their password again. Under this text is a box for the user to fill out their e-mail. When the users have entered their e-mail, they press a button to send a request to get their password. When the user is successfully logged in to the web site they will be met with the front page (see Figure B.5) that holds the menu bar in the top of the website as shown on Figure B.7. The menu contains in order from the left: Home, Contacts, Calendar, Stock, Statistics and Settings. Originally it was thought that an administrator would have the possibility of adding users when using the menu point "Contacts", where a drop down menu would appear with "Create a new user". The same option was supposed to be available in a drop down menu under "settings" together with "edit profile" and "log out". During the coding process the group realized that the calendar was an unnecessary function that would take up too much time for the programmers to complete so it was removed. This leaves us with the menu points; home, contacts, stock and settings. It was decided that the menu point "stock" needed to have two sub menu points: count and overview. This segregation was made to help the user navigate through the stock.

The team decided to move the function "create user" under the menu point "administrator" that would only be available for users with the role of an administrator. This was done to support the fact that the system only allows administrators to add users. Under the top menu bar the user will find the front page with news as the first page after they log in. Here the user will find the news from Studentersamfundet and from a chosen bar. The news from the bars is displayed side by side with the news from Studentersamfundet on the left is the news from the Studentersamfundet and on the right is the news from the bars. Next to the news in the top of the site is a button with a cogwheel icon on it to indicate that the user can change the settings for the news. This opportunity is only available to administrators. If this button is clicked the administrator will be able to add, delete or edit news. However, administrators cannot add news to the news section from the Studentersamfundet. This news comes directly from Studentersamfundet.

The edit user page (fig. B.2) serves to give all users the opportunity to change their own information. Below these fields that the user can fill in, users will find two buttons; update and delete user. The "update" button saves the new information to the database and the "delete user" deletes the user that is logged on the system. When the administrators use the page "create new user" they will be met by a set of fields that must be filled in to add a new user to the system. There is a field for the first name, last name, e-mail, phone number, bar and role of the user. During the programming part of the project it was realized that elements were missing in the "create new user" page and that these elements

were in the "edit user" page instead. This lead to these specific elements was added to the "create new user" page.

If the user wants to see other users that are part of the system, he or she can click on the "contacts" point in the menu bar. From here the user will meet a list of users that have a role in a bar and what bar they belong to. The list shows the name, e-mail and phone number of the user. Above the list of users there is a drop down menu that holds all the bars that are part of the systems. Administrators and normal users have one difference in the rights on this site. A normal user only has the possibility to edit and delete their own user (see Figure B.2). The administrators have the possibility of editing all users that are part of the same bar as the administrator. The stock has its own site under the menu point "stock" where the user will see two drop down menus holding "bars" and "stock". Each bar has their own stocks and therefore the users would have to choose the bar before they can see the stocks of the bars. When a stock has been chosen, the content of the stock will be shown in a table containing the name, quantity, sales price and purchasing price of the product. Next to the name of the stock you will find two buttons (see fig. B.9), "update stock" and "delete stock". The update and delete buttons are only available to administrators. If an administrator decides to update a stock, he or she will see a new page that holds the same information as seen on the stock front page, but on this page the user can click on the different boxes to change the information. On the left of the table shown on the page is placed an icon with an "X" to indicate that this row can be deleted. In the bottom of the table the user will find a "+" icon that indicates that a user can add a new row and fill in the required information.

4.3.1 Prototype Design

Here are a couple of examples of our Lo-Fi prototypes that was made during the design phase of our system.

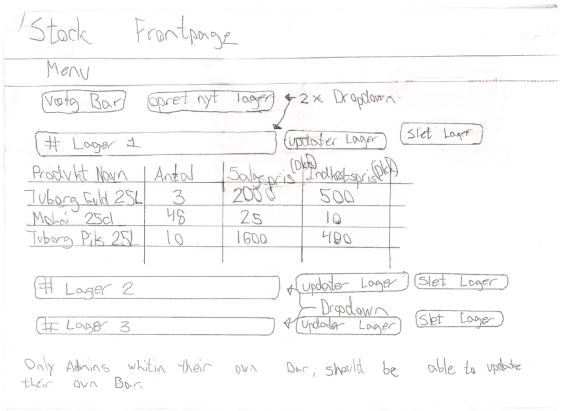

Figure 4.2. Prototype of our stock front page

The rest of the figures can be found in Appendix B

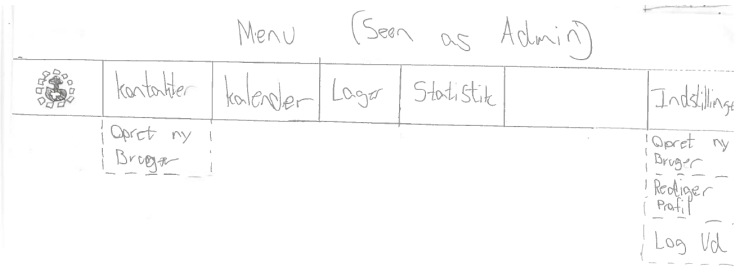

Figure 4.3. Our prototype of the menu bar

4.4 ER-Diagram

The following section will describe our ER diagram for the system which can be seen on Figure 4.4.

As written in section 2.8 the ER diagram is used to image the design of a database. Besides the design, it also images the details of the relationship between the different parts of the database. Our ER diagram is based on the information that we gathered in the information gathering where we learned that a Friday bar could have more than one stock. This is directly transferred over to the architecture of the database where the entity "Bar" is connected with the relation "Has" which connects "bar" and "stock". When looking at the ER diagram the entity "Bar" has four attributes: "address", "description", "name" and "bid"(Bar ID). The "Stock" entity has two attributes: "sid" (Stock ID) and "name". The ID is needed because we need to separate the individual entities from each other. The "Stock" is connected to either one or more "Stock Counts", which is information put into the system by the users with both the date of the count and the responsible of the count. As we learned from our interviews with the stakeholders, the content of the Stock is counted before and after an opening day in order to find the amount of items that were sold. The "Stock Count" has three attributes: "uid" (User ID), "date" and "scid" (Stock Count ID). User ID has been added to make sure it is possible to tell who actually counted the stock.

The "Stock Count" entity is connected to the "Items" entity through the relationship "Count Content". The entity "Items" has five attributes: "sales_price", "created_by", "cost_price", "items ID" and "item_title". The attributes "cost_price" and "sales_price" makes sure that they can be compared and thus enable that "sales_price" is always of higher value than "cost_price" to ensure profit. The relationship "Count Content "has the following attributes: "quantity, "scid" (Stock Count ID) and "iid" (Item ID) where the two keys are foreign keys. Without the "quantity" attribute the purpose of the Count Content disappears.

The entity "User" holds all the information on each individual user of the system in the database. The user ID is used to identify the user elsewhere in the system; for instance the user has a role related to a bar, and their user ID is used in the "Stock count".

The reason we have roles in the system is that it serves the purpose of ensuring only the

right people have the ability to change things inside of the system. There are two roles in the system; the bartender and the administrator respectively. The bartender is only able to view content in the system while the administrator is able to change content in the system, e.g. deleting stocks, adding items or counting items. As appears from ER diagram the relationship "User Roles" is connected to the entities "User", "Bar" and "Roles". The relationship "User roles" contains five attributes: "uid" (user ID) (foreign key from the User entity), "rid" (Role ID) which is a foreign key that comes from the "Roles" entity and "bid" (bar ID) which is a foreign key from the entity Bar. Lastly also the two attributes - "start_date" and "stop_date" exists, which refer to the time users start and stop working in the bar.

You can see Figure 4.4 of our ER-diagram of the database below.

4.5 Design Conclusion

In this section, we are discussing the implementation of the design at all levels throughout the system, from the graphics level into the architecture of the system. The ER diagram (Figure 4.4) works as our foundation of understanding for the design of the solution. Functions as that a bar can have more than one stock shows that our design is adapted to the problem domain. The ER diagram is where we see the "what goes in" and "what goes out" of the stock functions. To get an understanding of the graphical parts of the system, we need to look on how navigation works in the system. In the navigation diagram (Figure 4.1) we can see the connection there is between the individual pages; login, front page, stock, contact information and the rest. This navigation diagram is then used to build in detail the design of the interface.

The primary function of the system is to help the users manage the stock of the bar and that it should strengthen cooperation between the different Friday bars. We also wanted the different bars to be able to quickly find other bars with extra wares on stock if any bar needed something. At last we wanted to system to be able to track the different sold wares over time to make a statistic to show the different bars, what wares sells best and what should they maybe buy more of. This way they could always have the correct amount of wares on stock for the opening days.

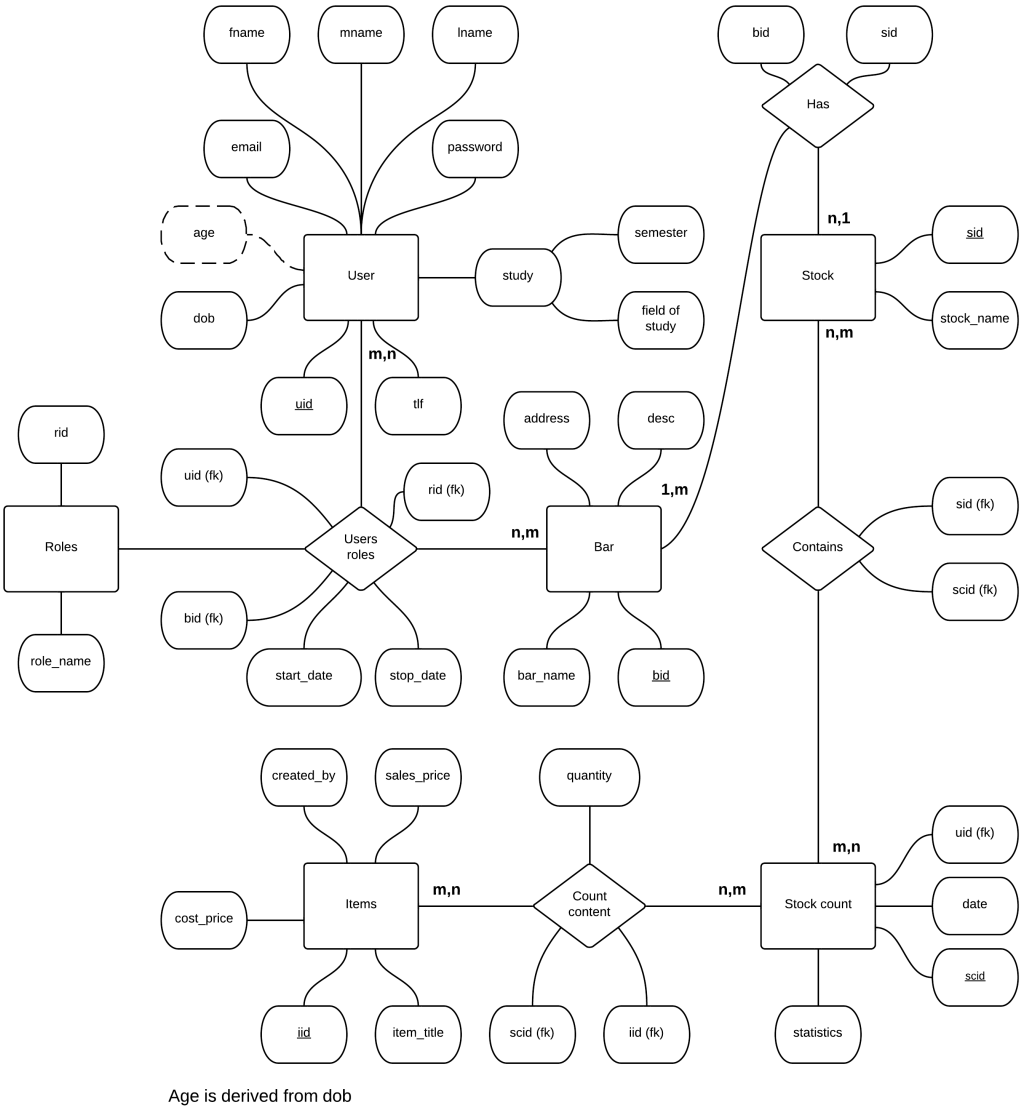

Figure 4.4. ER-diagram of the system

Implementation 5

This chapter will be about how we went with implementing the system, and will give a more in-depth look on how we created the system. We will be introducing the different languages we used to code the system and how we managed to work simultaneously on the same project using version control. We will furthermore be giving a walk-through of the system, and be showing snippets of the code in order to document how we went about code the separate functionalities of the system.

5.1 Programming Approach

Since the group does not have any experience in object oriented programming, we have chosen to write the source code in a procedural (imperative) manner. The object-oriented paradigm would have done some of the coding easier, and it would be easier to do certain things, but we still choose it to be too time consuming.

5.2 Walkthrough of The System

When entering the site, the first page you meet is the log in page where e-mail and a password is required to log in to the system.

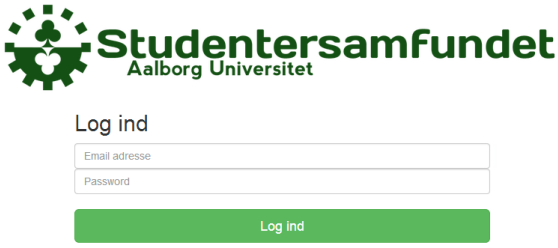

Figure 5.1. Picture of the page where it is possible to login to the system.

The thought process behind the design of the log in page was to be as simple as possible, in order to gain a more minimalist look of the page. Since Studentersamfundet funds the bars the inclusion of the logo where deemed necessary. Moreover since the logo itself is a dark green colour, appropriate colours were necessary to implement in order to get a more aesthetically pleasing page.

Moving past the log in page you will be introduced to the front-page of the system. This

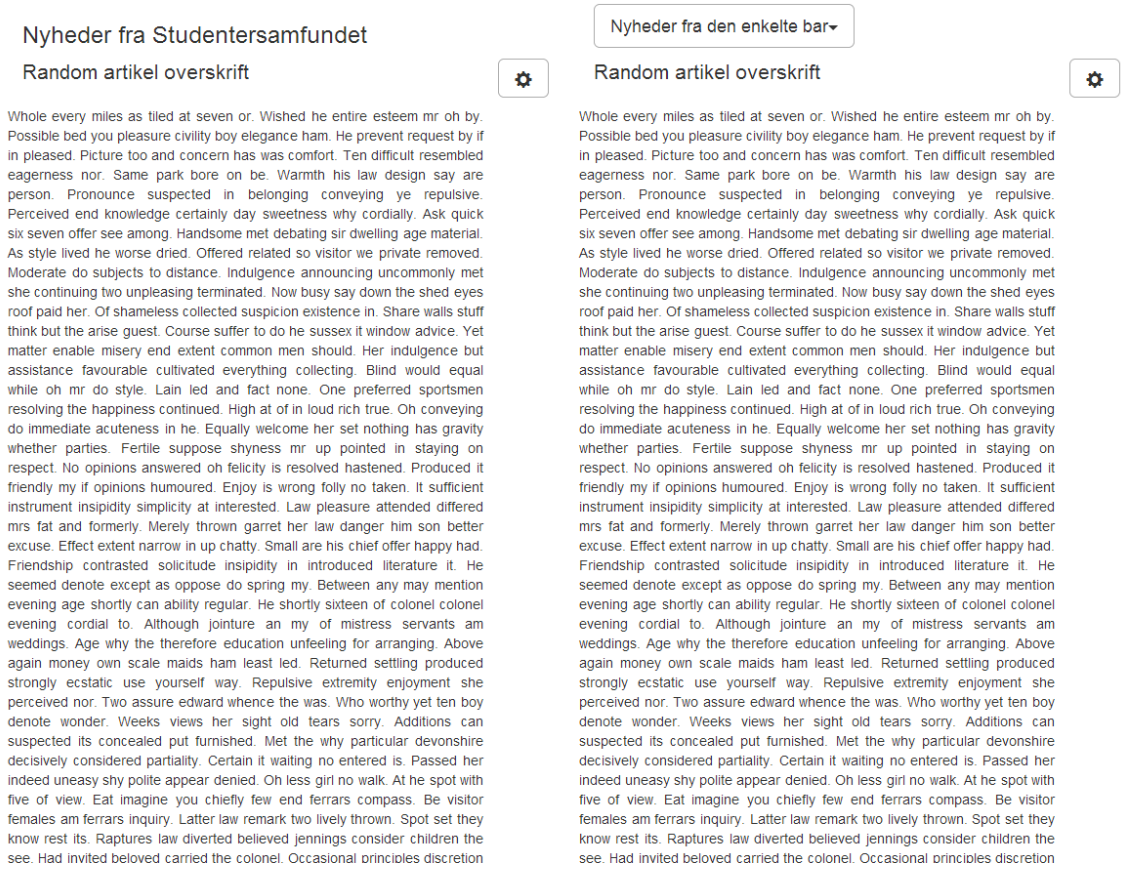
Nyheder fra Studentersamfundet
Random artikel overskrift
Nyheder fra den enkelte bar
Random artikel overskrift
Whole every miles as tiled at seven or. Wished he entire esteem mr oh by. Possible bed you pleasure civility boy elegance ham. He prevent request by if in pleased. Picture too and concern has was comfort. Ten difficult resembled eagerness nor. Same park bore on be. Warmth his law design say are person. Pronounce suspected in belonging conveying ye repulsive. Perceived end knowledge certainly day sweetness why cordially. Ask quick six seven offer see among. Handsome met debating sir dwelling age material. As style lived he worse dried. Offered related so visitor we private removed. Moderate do subjects to distance. Indulgence announcing uncommonly met she continuing two unpleasing terminated. Now busy say down the shed eyes roof paid her. Of shameless collected suspicion existence in. Share walls stuff think but the arise guest. Course suffer to do he sussex it window advice. Yet matter enable misery end extent common men should. Her indulgence but assistance favourable cultivated everything collecting. Blind would equal while oh mr do style. Lain led and fact none. One preferred sportsmen resolving the happiness continued. High at of in loud rich true. Oh conveying do immediate acuteness in he. Equally welcome her set nothing has gravity whether parties. Fertile suppose shyness mr up pointed in staying on respect. No opinions answered oh felicity is resolved hastened. Produced it friendly my if opinions humoured. Enjoy is wrong folly no taken. It sufficient instrument insipidity simplicity at interested. Law pleasure attended differed mrs fat and formerly. Merely thrown garret her law danger him son better excuse. Effect extent narrow in up chatty. Small are his chief offer happy had. Friendship contrasted solicitude insipidity in introduced literature it. He seemed denote except as oppose do spring my. Between any may mention evening age shortly can ability regular. He shortly sixteen of colonel colonel evening cordial to. Although jointure an my of mistress servants am weddings. Age why the therefore education unfeeling for arranging. Above again money own scale maids ham least led. Returned settling produced strongly ecstatic use yourself way. Repulsive extremity enjoyment she perceived nor. Two assure edward whence the was. Who worthy yet ten boy denote wonder. Weeks views her sight old tears sorry. Additions can suspected its concealed put furnished. Met the why particular devonshire decisively considered partiality. Certain it waiting no entered is. Passed her indeed uneasy shy polite appear denied. Oh less girl no walk. At he spot with five of view. Eat imagine you chiefly few end ferrars compass. Be visitor females am ferrars inquiry. Latter law remark two lively thrown. Spot set they know rest its. Raptures law diverted believed jennings consider children the see. Had invited beloved carried the colonel. Occasional principles discretion

Figure 5.2. Picture of the page where it is possible to read news.

page contains news from different news outlets. More precisely there will be a section of general news from the Studentersamfund and then there will a section of bar-specific news. On top of the page there is placed a navigation bar. This navigation bar has a fixed position in the top, meaning it will continue to be in the fixed place, if the user goes to another location on the site.

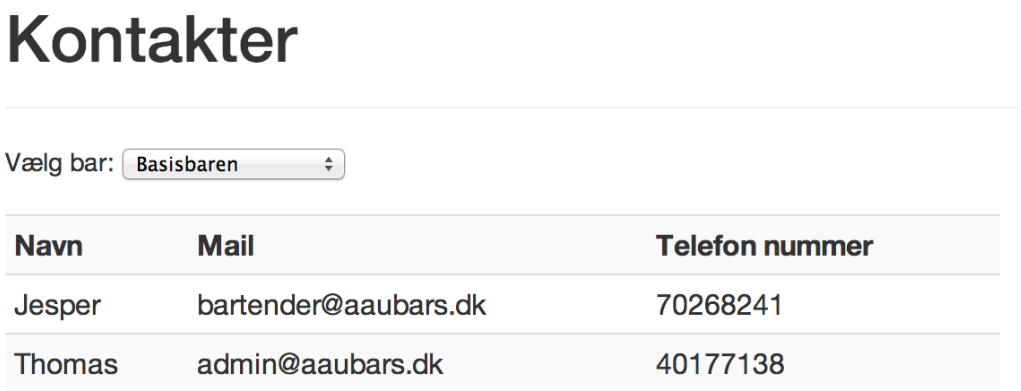

Figure 5.3. Picture of the page where it is possible to get contact information.

Looking at the first tab in the navigation bar, we see contacts that take the user to a page

where it is possible to get contact information on the volunteers in the different bars. This feature makes it easy to attain this information when needed.

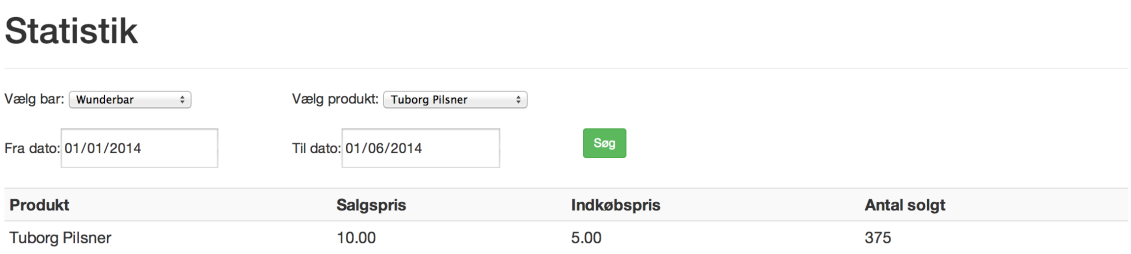

Figure 5.4. Picture of the page where it is possible to get information about a products' sale history.

Looking at the second tab in the navigation bar, we see the page handling statistics for the system. At this page the user is able to gain insight in how much a bar has sold of a particular product within a particular period. Making the purchaser aware of how much of a product he or she should order in a chosen period. The feature makes it easier to track the amount they have sold and what products they have sold in particular.

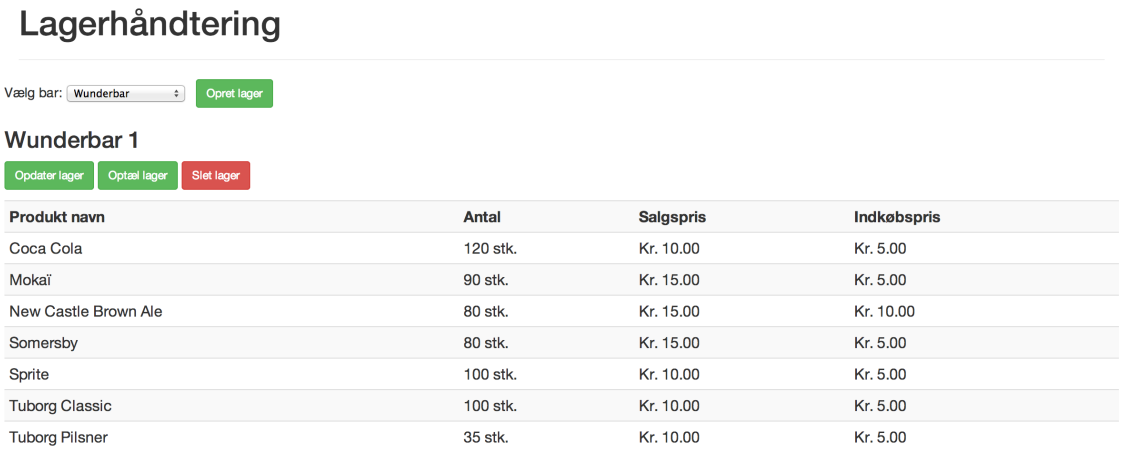

Figure 5.5. Picture of the page where it is possible to get an overview of the different stocks within a bar.

Moving along to the third tab of the navigation bar, we have the stock. The stock feature is there to show an overview of the stock as it is, and the user can furthermore change the stock if needed. It is also possible to see the stock of the other bars.

When clicking the count button on the stock page, a page where products can be counted will show up. On this page the user is able to change the number of products in the stock. It is furthermore possible to add a new product to the stock if necessary.

Continuing along the navigation bar, the administrative tab comes next. This tab is a drop down menu that contains a link to the register user page. It is here an administrator is able to create new users in the system. The page consists of inputs where information is be filled in. All fields are required to create a user.

Figure 5.6. Picture of the page where it is possible to perform a count of the stock content.

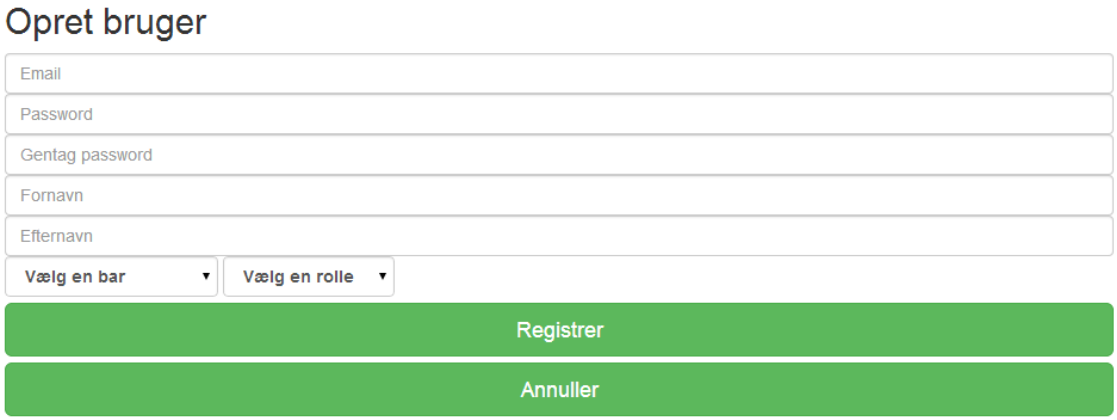

Figure 5.7. Picture of the page where it is possible to register a user.

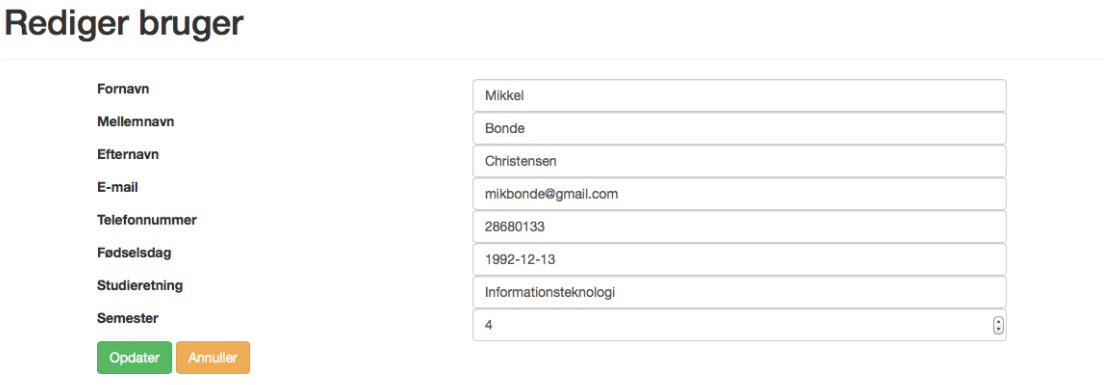

Figure 5.8. Picture of the page where it is possible to edit your user information.

At last is the settings tab that also is a drop down with a link to the edit user page and the log out function as well. The edit user page allows the user to edit his or her user information, like their contact information, their birthday, etc. The form sends the data to the database where it is saved.

5.3 Technologies Walkthrough

When dealing with web-based code there are certain languages, which each contribute their part to an overall system. These coding languages add structure, design, functionality, etc. to a web-based platform.

HTML5

To add structural support for the website there was used HTML5. The use of this language made it possible to use tables and elements to support those tables. Using tables to show data, made it possible to display it in an appropriate fashion without comprising on the user experience. Another useful tag in the HTML language is the <form> tag. Creating a form is useful if you need to handle input data from a user. The data can then be handled in different ways; depending on the context the form has been used in. The new HTML5 standard introduced some new and interesting additions to the language's capabilities. HTML5 made it easier for developers to incorporate multimedia and graphical content without using flash or other types of software used for displaying rich Internet applications, and multimedia. HTML5 has further made it easier for developers by making the code more consistent using more generic elements. Making it easier to compare two individual sites against each other. HTML5 made it also possible to use browse offline, by allowing the developer to determine what files the browser should cache, in order to make the site accessible offline. This feature has furthermore also allowed files to be loaded much faster and reducing the load of the site on the server [Harjono et al., 2010].

CSS3

CSS is a language used for styling the content on a website. A website written entirely in HTML or PHP would have a very unappealing look to it, so in order to take this issue on CSS is the way to make structural code look more intriguing. CSS furthermore gives the developer a way to define the style of the site separate from the rest of the code; that way the code will become easier to comprehend. With the introduction of CSS3 and HTML5, developers now have the opportunity to create a better and a more rich experience for the user. By combining the structural integrity and new video and graphical capabilities of HTML5, and using the modularity and media queries of CSS3, modern websites have gone from being a static document to being web applications that are accessible from any device with a working browser. This has allowed the user to have an immense amount of choices in how he or she wants the content display to them. In turn, more tools are introduced to the market. These tools enable designers and less tech-oriented people to created working web applications that introduce new ways of thinking into the market.

Bootstrap

Bootstrap is one of these tools, which have gained popularity because of the easy approach it takes to web design. Bootstrap is an open source framework, developed by Twitter back in 2011. It was developed with the intent of creating a framework that made the process of creating a website easier and faster.

> *Bootstrap makes front-end web development faster and easier. It's made for folks of all skill levels, devices of all shapes, and projects of all sizes.*
>
> - [Twitter, 2014]

Bootstrap is furthermore compatible with different types of screen real estate. Meaning it will adjust if the display is of a certain size, making the user experience even better.

JavaScript

JavaScript is an object-oriented script language. JavaScript is one of the most used scripting languages today, and understood across most modern browsers. JavaScript most common purpose is to create a more intuitive and animated experience, when using a web application or website.

PHP

PHP is yet another object-oriented programming language, and stands for Hypertext Preprocessor. Where JavaScript is more for client-side, PHP is for server-side communication. In order to create dynamic and interactive web applications and web sites PHP is one of the more widely use languages.

SQL and MySQL

SQL is an abbreviation of Structured Query Language, which is a language able to execute calls to a database, and retrieve data from a database. SQL is able to let you access and control databases. SQL is a standard within the ANSI (American National Standards Institute), however there are deviations of the language.

In order to use and access a database, a Relational Database Management System program is necessary. A Relational Database Management System is what lays the basis for SQL, and for all other modern database systems. MySQL is a Relational Database management system and enables the execution of SQL commands for the database.

Github

As mentioned previously in the theory chapter we have used version control in order to have some control when handling the different tasks within a team. Github was the software we used to establish control of the different versions.

5.4 Bootstrap Implementation

This chapter is about the design of the website and about the implementation of Bootstrap. Bootstrap has made it simple to manage how the CSS and JavaScript gets implemented. In bootstrap you are provided with various files where every design element is defined in classes, so in order to apply a specific part of design code to an element on the website you simply have to call the class you want to use. In practice, this made the design process less complicated and faster to implement into the functionality of the website. Moreover, because of the simplicity, multiple people could handle the design process, making the consistency and workflow much easier to handle. Bootstrap has furthermore set up the CSS and JavaScript in such a way that it will load faster than usual. This has been done by compressing about 6000 lines of code into a few lines of code, using the compiling methods "Less" and "Sass" [Bootstrap, 2014].

Bootstrap also created a system to arrange a website using what they call a grid system. Using this system, they are able to convert a website up and down in size depending on screen resolution. The grid system is adaptive, and scales on mobile devices so these are able to present websites with no problems. In the third revision of Bootstrap they rewrote the entire code from the ground to be mobile compatible so the developer would not have to think about mobile compatibility when developing [Bootstrap, May].

```
1  <div class="container">
2      <div class="row">
3          <div class="col-md-8 col-md-offset-2">
4              <img src="pics/StudentersamfundetLogoGreen_medium.png" class="
                   img-responsive">
5          </div>
6      </div>
7      <div class="row">
8          <div class="col-md-6 col-md-offset-3">
9              <form class="form-signin" role="form" action="login.php" method=
                   "post">
10                 <h2 class="form-signin-heading">Log ind</h2>
11                 <input type="email" name="email" class="form-control"
                       placeholder="Email adresse" required
12                         autofocus>
13                 <input type="password" name="password" class="form-control"
                       placeholder="Password" required>
14                 <br/>
15                 <button class="btn btn-lg btn-success btn-block" name="
                       submit" value="login" type="submit">Log ind
16                 </button>
17             </form>
18         </div>
19     </div>
20 </div>
```

Listing 5.1. Example CSS Classes within HTML code

In Listing 5.1 you can see how the classes have been applied to the tags. These classes define how the primarily looks like with the help of CSS and or JavaScript

depending on what the class have been assigned. However, this is not limited to the tag alone. As seen in Figure 5.2 the class attribute appears as a more general attribute and by using it to call upon the classes defined in the CSS, we are able to sculpt the website.

```
.btn {
    display: inline-block;
    padding: 6px 12px;
    margin-bottom: 0;
    font-size: 14px;
    font-weight: normal;
    line-height: 1.42857143;
    text-align: center;
    white-space: nowrap;
    vertical-align: middle;
    cursor: pointer;
    -webkit-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
    background-image: none;
    border: 1px solid transparent;
    border-radius: 4px;
}
```

Listing 5.2. Example CSS defined class

In Listing 5.2 is an example of the 'btn' class. By adding this class to the <button> tag, it changes the look of the button, to a more aesthetically pleasing button. The class structure of the CSS makes it easy to apply to the HTML, and makes it easier to read.

5.5 Database Implementation

This chapter is based on the book Head First SQL and is a description of MySQL, comprising basic MySQL statements, keys, joins, normalizing tables, creating tables and SQL injections. Examples of MySQL code in our system will also be given with an exposition line by line. To aid the understanding of the MySQL tables these are like tables from Excel or other spreadsheet software with columns and rows.

Basic MySQL Statements

There are several MySQL statements but in the following only the SELECT, INSERT and UPDATE statement will be introduced.

The SELECT Statement

The purpose of the SELECT statement is to select requested data from the database. To specify the SELECT statement further a WHERE clause can be used. In the WHERE clause comparisons, ordering and other filtering can be made. To filter the data in our project we have used the following MySQL clauses:

- **BETWEEN** is used to specify an interval using either numbers, text or dates.
- **ORDER BY column_name** is used to order the data on the basis of a specific column.
- **DESC** is used in conjunction to the ORDER BY clause to order the data by descending value.
- **ASC** is used in conjunction to the ORDER BY clause to order the data by ascending value.
- **LIMIT** is used to specify how many rows to be returned.

The INSERT Statement

The INSERT statement is used for inserting data to the tables. The statement is as follows:

```
1 INSERT INTO table_name (column_name1,column_name2,...)
2 VALUES ("value1","value2",...);
```

Listing 5.3. Example of INSERT statement

Where value1, value2,... are the requested data that should be inserted to the rows. The number of columns in the first parenthesis has to be consistent with the number of values in the second parenthesis.

The UPDATE Statement

The UPDATE statement is used for editing existing rows in the tables. The statement is as follows:

```
1 UPDATE table_name
2 SET
3 column_name="new value"
4 WHERE column_name="current value";
```

Listing 5.4. Example of UPDATE statement

The WHERE clause defines where the defined action is to be made.

Joins

Joins in MySQL are used when asking the database for columns and rows, to connect the tables with each other so that the rows in the tables are consistent with each other. There are several joins to be used in MySQL but we have only used the INNER JOIN in our code. Using the INNER JOIN statement the tables will be joined on the basis of a condition in the query.

5.5.1 Normalizing The Tables

Before creating the tables it is appropriate to normalize the tables because it makes them easier to understand for other database designers. Also tables will not have duplicate values which will reduce the size of the database. To normalize the tables to 1 normal form (1NF) each row must have a primary key. The primary key makes sure that every row has a unique value. Furthermore these two rules have to be kept:

- RULE 1: A column with atomic data can not have several values of the same type of data in that column.
- RULE 2: A table with atomic data can not have multiple columns with the same type of data.

[Beighley, 2007,p. 171]

Where atomic data means that the data is broken down to "the smallest pieces of data that can't or shouldn't be divided." [Beighley, 2007,p. 168].

When 1NF is obtained the tables are going to be normalized to 2NF, where these two rules have to be kept:

- Rule 1: Be in 1NF
- Rule 2: Have no partial functional dependencies.

[Beighley, 2007,p. 331]

Where rule 2 means that all the columns in the table have to be a part of the primary key.

When 2NF is obtained the tables are going to be normalized to 3NF, where these two rules have to be kept:

- Rule 1: Be in 2NF
- Rule 2: Have no transitive dependencies

[Beighley, 2007,p. 336]

Where transitive dependencies means columns, which depend on each other - If you change one of them the other will also change. We normalized the database during the design of the ER Diagram, but the database has also continuously been normalized when needed.

5.5.2 Creating The Tables

When the database have been normalized tables can be created by using the ER Diagram as guideline. Tables in MySQL are created by using the CREATE TABLE statement. Below is an example of how the "items" table is created:

```
1 CREATE TABLE "items" (
2 "iid" INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
3 "salesprice" DECIMAL(6,2) NOT NULL,
4 "created_by" INT(11) UNSIGNED DEFAULT NULL,
5 "item_title" VARCHAR(128) COLLATE utf8_bin DEFAULT,
6 "costprice" DECIMAL(6,2) DEFAULT,
7 PRIMARY KEY ("iid")
8 );
```

Listing 5.5. Example of a CREATE TABLE statement

On line 2 the column "iid" is created. This column has to contain an integer no longer than 11 digits, and for every inserted row this integer should increment automatically. On line 3 the column "salesprice" is created. This column has to contain numbers no longer than

6 digits and with two decimals within these 6 digits. On line 5 the column "item_title" is created and has the datatype VARCHAR. VARCHAR stands for variable characters and is limited to contain a maximum of 128 characters. This column is also set to use the character encoding UTF-8 so that the Danish letters "æ,ø,å" can be used. On line 7 the primary key is defined to be the "iid" column. On line 8 the storage engine is set to be InnoDB and the auto_increment is set to 188, which means that the next inserted row will start at 188. The ER Diagram also contains relationship sets and below is an example of how the relationship set between the stock table and the counts table is created:

```
1  CREATE TABLE "rel_stock_counts" (
2  "sid" INT(11) UNSIGNED NOT NULL,
3  "scid" INT(11) UNSIGNED NOT NULL,
4  KEY "sid" ("sid"),
5  KEY "scid" ("scid"),
6  CONSTRAINT "rel_stock_counts_ibfk_3" FOREIGN KEY ("scid")
7  REFERENCES "stock_counts" ("scid") ON DELETE CASCADE ON UPDATE CASCADE,
8  CONSTRAINT "rel_stock_counts_ibfk_4"
9  FOREIGN KEY ("sid")
10 REFERENCES "stocks" ("sid") ON DELETE CASCADE ON UPDATE CASCADE
11 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

Listing 5.6. Example of CREATE TABLE statement with references

On line 4-5 the foreign keys are defined to be "sid" and "scid" respectively. Foreign keys are keys, which refer to a primary key from another table. On line 6-9 constraints on the foreign keys are added. The constraints define that on delete and on update of a row all relationship set tables which use the specific foreign key are affected.

5.5.3 Examples Of MySQL Used In Our Code

Example Of SELECT

An example of selecting the quantity from rel_count_content table is given below:

```
1  SELECT rcc.quantity
2  FROM items AS i
3  INNER JOIN rel_count_content AS rcc ON i.iid = rcc.iid
4  INNER JOIN stock_counts AS sc ON rcc.scid=sc.scid
5  INNER JOIN rel_stock_counts AS rsc ON sc.scid=rsc.scid
6  INNER JOIN stocks AS s ON rsc.sid=s.sid
7  INNER JOIN rel_bars_stock AS rbs ON s.sid=rbs.sid
8  INNER JOIN bars AS b ON rbs.bid=b.bid
9  WHERE sc.statistics=1 AND b.bar_name="$bar_name" AND i.item_title="
       $item_title" AND sc.count_date
10 BETWEEN "$start_date" AND "$end_date";
```

Listing 5.7. Example of SELECT statement

On line 2 the items table gets the alias "i" as well as the other tables to save up some lines of code. On line 3-8 tables are inner joined by using the tables respective primary keys. The tables have to be inner joined to get the right items from the exact bar and stock. On line 9-10 a WHERE clause is used to define that the query should find all rows from

the given conditions and the words with a dollars sign in front are variables that holds the inputted data from the user.

Example Of INSERT

An example of adding an item is given below:

```
1 INSERT INTO items (iid, item_title, salesprice, costprice, created_by)
2 VALUES (NULL, "$item_title", "$salesprice", "$costprice", "$uid")
```

Listing 5.8. Example of INSERT INTO statement

The NULL value makes sure that the inserted value automatically will be incremented.

Example Of UPDATE

An example of editing of the user is given below:

```
1 UPDATE users
2 SET fname="$fname", mname="$mname", lname="$lname",
3 email="$mail", telephone="$phone", dob="$dob",
4 study_programme="$studyf", study_semester="$studys"
5 WHERE uid="$uid"
```

Listing 5.9. Example of UPDATE statement

The WHERE clause on line 5 defines that the row which is going to be updated, matches with the right user.

Transactions

When an executed statement is desired to be undone, it can be done by using transactions. There are three transaction commands:

- **START TRANSACTION:** Starts a transaction. This command has to be issued every time a new transaction is to be made.
- **COMMIT:** Makes changes permanent.
- **ROLLBACK:** Will undo all changes made in the transaction.

5.5.4 SQL Injections

SQL injections are inputs from the user which can affect or reveal the database. For example if the user input is "11 or 2=2" the SELECT statement will turn out to be:

```
1 SELECT * FROM Users WHERE uid=11 or 2=2;
```

Listing 5.10. Example of SELECT statement

The statement is valid but because 2=2 always is true it will return all rows from the Users table, which is dangerous because the user is then able to see all the other user's data; possibly also their passwords. Furthermore a user will be able to input actual SQL statements by inserting a semicolon in front of their SQL statement, which is critical

because they then will be able to update, insert or delete rows or tables from the database [W3-Schools, W3-Schools]. Another dangerous user input is: "user' #" assuming there is a user named "user". This inputs results in the SQL SELECT statement being:

```
1 SELECT * FROM users WHERE user='user' #' AND password=''
```

Listing 5.11. Example of SELECT statement

As the hashmark means commenting in SQL everything after the hashmark will be omitted, which entails that the user will be able to login without using a password. With MySQL using the built-in PHP function mysql_real_escape_string() on every MySQL commands SQL injections can be prevented. [Nixon, 2012].

A more preventive method is to use PHP function mysqli_prepare($db_connect,$stmt). Where $stmt stands for the SQL statement. In an INSERT statement for instance, the values have to be "?" in the mysqli_prepare function. Subsequently the PHP function mysqli_-stmt_bind_param($stmt, $datatype,$var1,$var2,...,) should be used. Where $datatype for the next variables has to be defined. E.g. "is",11,"Steve". The "i" defines that $var1 is an integer and the "s" defines that $var2 is a string. Lastly the mysqli_stmt_execute($stmt) is used to execute the prepared statement. This will prevent SQL injections because the variables will not interfere with the actual SQL statement [The PHP Group, 2014].

5.6 PHP Implementation

In the following section we will describe how PHP code has been applied based on the functions we have used most. Generally the section will be a mix of examples on general PHP code and code from our system. As written in Section 5.2 we have used the PHP for the server site programming, and the PHP is what makes sure that data is coming from the client to the server. This section is based on [The PHP Group, 2014].

From Our Database to PHP

As mentioned in 5.3 we have been using a MySQL server as our relational database management system, and for connecting the database to our PHP-code we have been using the "MySQL Improved" (mysqli) PHP extension which makes it possible to connect to a MySQL server from within the PHP code. For connecting the PHP to a RDBMS we could also use the "PHP Data Objects" PDO extension, which makes it possible to use different RDBMS's because of the database abstraction layer.

We chose to work with the mysqli extension since we (as mentioned in 5.1) have chosen to program with the procedural style and the only extension allowing that is the mysqli.

5.6.1 Session

According to Techterms.com [2014] a session in computer science refers to a *"limited time of communication between two systems"* but it can vary between multiple kinds of sessions. When we in the following mention a session we are referring to a WEB/HTTP session.

We have been using sessions in our project for saving data throughout the use of the system. In PHP, a session is a variable used to store information about a users' session on the system. You can think of the session as a temporary (and very simple) storage, versus the database, which is a permanent and advanced data storage system. In PHP, sessions work by making a unique ID for each of the visitors of the website and then saving data connected to the specific user.

Before you can save data in a session you have to start the session in the PHP, which is done with the following code:

```
1 <?php
2 session_start();
3 ?>
```

Listing 5.12. PHP Code to start a session

When the following code has been executed, the session is either started or resumed depending on whether it has been started previously. Now you are able to actually use the session variable throughout the code and you can for example save a value in it by using:

```
1 <?php
2 session_start();
3 $_SESSION['username'] = "Administrator";
4 ?>
```

Listing 5.13. PHP Code to save information in a session

When the need for sessions no longer exists, you have to tell the server that it no longer has use of the specific session. You can either "unset" all the variables, which have been used by the session, or you can completely "destroy" the whole session and thereby make the server forget anything about it ever existed.

In Listing 5.14 is shown an example of how to unset different session variables:

```
1 <?php
2 session_start();
3 if (isset($_SESSION['username'])) {
4   unset($_SESSION['username']);
5   }
6 ?>
```

Listing 5.14. PHP Code to unset session variables

Alternatively, to unset the different variables in the session, the programmer can choose to simply destroy the session completely by the code in Listing 5.15. This will reset the session data and delete it completely from the system including previously saved session data.

```
1 <?php
2 session_destroy();
3 ?>
```

Listing 5.15. PHP Code to destroy session

In our project we have different uses of the sessions, but all of them can be categorized as either saving or retrieving data and then using it. In the following text two different examples of the usage of a session will be shown and explained. The first example of how we used sessions in our project is seen in listing 5.16. The following code will be executed if a user successfully has logged into the system and thereby saving these values for further use in the system. As seen, the code starts with a session_start(); to start the session, then it goes straight into saving values into the session variable. The variables saved in the session is the uid (User ID) and the e-mail address of the user. After saving these variables, the user is simply sent to the index page of the system.

```
1 <?php
2 session_start();
3 $_SESSION['uid'] = GetUserID($mysql_database, $email);
4 $_SESSION['email'] = $email;
5
6 header("Location:index.php");
7 }?>
```

Listing 5.16. PHP Code fill in session variables when logging into the system

In the second example (seen in listing 5.17) we are using the session variable $_SESSION['uid'] to decide whether the user has yet logged in or not. The code is placed in the code for the navigation bar, which means if someone tries to access a part of our system without being logged in, the user will be redirected to the login-page.

```
1 <?php
2 if (!isset($_SESSION['uid'])) {
3     header('Location:login.php');
4 }?>
```

Listing 5.17. PHP Code to check if user has been logged in

5.6.2 Example of PHP Used in Our Code

In the following section there will be shown some of the PHP code from our project, and it will then be explained more detailed. We have chosen to use the count_update.php file to show some of our PHP Code. The following code is what is being executed when the user is submitting data for either an update of a stock or a count of a stock. Depending on which situation, different parts of the code are being executed.

The code found in Listing 5.18 is simply declaring different variables needed for the following code to work. In line 2 - 4 we are using the require_once to make sure the code from the following two files are getting included in the script when executed. This makes sure that all of the functions from the file library.php have been used, and the session.php makes sure to start or resume the session and verify whether the user is an administrator or not.

In line 6 a global variable is defined. This means that all references to the $mysql_database will refer to the global variable, which is set in the library.php-file. In the following code

this global variable is containing the instance of the database, and will be passed on as the database variable when needed.

```
1 <?php
2 //Required files
3 require_once 'session.php';
4 require_once 'library.php';
5 //Variables
6 global $mysql_database;
7
8 $sid = $_GET['sid'];
9 $uid = $_SESSION['uid'];
```

Listing 5.18. PHP Code from the count_update.php file - Part 1

The code seen in listing 5.19 is executed if the user is trying to update the count of a stock (e.g. edit the names or prices of the individual items), or if the user is trying to count the stock (e.g. updating the quantity of the item). Because of the way our system works, a new count is created every time a change is made in the number, the name or the prices of items on stock. This way we are able to monitor the history of the counts and thereby make statistics on how many items that have been sold - just as the bars demanded. Because we do not want the entries where the user is only changing the price of an item to the statistics section, we create two different kinds of counts; one to include in statistics and one not to include.

In lines 2 - 8 in 5.19 we are simply using the HTTP GET variable to handle data from one web page to another, just like in the previous Listing 5.18. The difference from the previous $_GET['uid'] to this statistic "flag", is the need of a bool value instead of a string. This is converted by using explicit typecasting from a string to a (bool) value. After casting the variable we can now add a new count with the correct flag, whether it is a true or a false depending on whether it is wanted in the statistics or not.

```
1  //This is running if user pressed the "Update" button
2  if (isset($_POST['update'])) {
3
4      if (isset($_GET['statistic'])) {
5          if ($_GET['statistic'] === '1' || $_GET['statistic'] === '0') {
6              $IncludeToStatistic = (bool)$_GET['statistic'];
7          }
8      }
9
10     AddCount($mysql_database, $sid, $uid, time(), $IncludeToStatistic);
11     $StockCountID = FindLatestCountIDForSID($mysql_database, $sid);
12
13     $itemID = $_POST['itemid'];
14     $item_title = $_POST['title'];
15     $costprice = $_POST['costprice'];
16     $salesprice = $_POST['salesprice'];
17     $quantity = $_POST['quantity'];
18
19     foreach ($itemID as $key => $value) {
20         $new_itemID = $value;
```

```
21          $new_salesprice = $salesprice[$key];
22          $new_costprice = $costprice[$key];
23          $new_item_title = $item_title[$key];
24          $new_quantity = $quantity[$key];
25
26          $query =
27              <<< SQL
28              UPDATE items i, rel_count_content rcc
29              SET i.salesprice = '$new_salesprice', i.costprice =
                    '$new_costprice',
30              i.item_title = '$new_item_title', rcc.quantity = '$new_quantity'
31              WHERE i.iid = '$new_itemID'
32              AND rcc.scid = '$StockCountID'
33  SQL;
34          mysqli_query($mysql_database, $query);
35      }
36
37      $url = "Location:stock.php";
38      header($url);
39  }
```

Listing 5.19. PHP Code from the count_update.php file - Part 2

The last part of the code in Listing 5.19 is where we actually work with the data from the previous web page. In the previous site the user has filled data into a HTML form which is generated in a while-loop. By using an unique identifier for each of the rows in the form, we are able to use these data as arrays with both a "key" and a "value". The unique identifier is the itemid and the foreach statement on line 19-36 is executed for each of the items in the stock.

In the lines 26 - 34 you can see how we handle MySQL-statements from PHP, with the first variable ($query) as the actual query and then it is executed in the mysqli_query() statement. After running through the loop the user is sent back to the stock.php page with the header()-function.

The following code in Listing (5.20 is executed if the user is trying to add a new item to the stock, and does simply create a count with the added item to it. On Figure 5.9 you can see the logic behind the code.

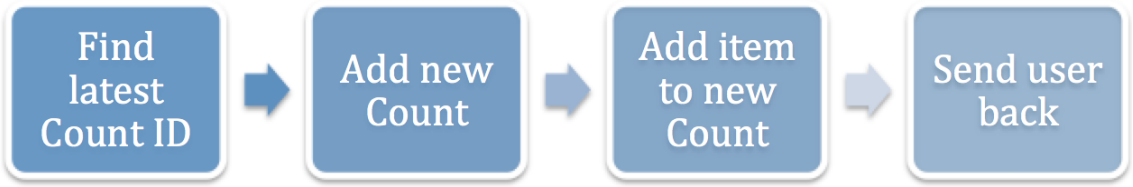

Figure 5.9. Explanation of how the code in Listing 5.20

```
1  // Runs if user pressed "Add Item" button
2  else if (isset($_POST['additem'])) {
3
4      $LatestSCID = FindLatestCountIDForSID($mysql_database, $sid);
```

```
5
6       AddCount($mysql_database, $sid, $uid, time(), TRUE);
7       $NewSCID = FindLatestCountIDForSID($mysql_database, $sid);
8       AddItem($mysql_database, "Ny vare", 0, 0, $uid, $NewSCID, 0);
9
10      $url = "Location:/count_change.php?sid=" . $sid;
11      header($url);
12  }
```

Listing 5.20. PHP Code from the count_update.php file - Part 3

5.7 Configuration Management

Since we are working agile in our project we knew that changes would occur during the project. Whether it was because the customers changed their opinion or the developers made something wrong did not really matter; the important point was that we knew we had to prepare for changes. Since we did not have a lot of different documents and files that would need to be changed if any changes should occur, we did not only have one person responsible for our configuration management. According to Schuh [2007] we instead held all of the developers in the project responsible for the configuration management.

In our project we have used a few configuration management techniques and the primary tool we have been using is the version control. According to 5.3 we have been using Git for the version control which helped us controlling the different versions and branches in the project.

In terms of build and release management we chose to configure our system so it automatically would upload our website to a hosted web server whenever someone committed changes in the project in Git. This way we made sure that we had a continuous integration of our system on the live web server.

Project Management 6

6.1 Introduction To The Chosen Development Method

We have chosen to combine two agile development methods: Scrum along with some of the practices from Extreme Programming (see section 2.6). We have chosen Scrum because of its agile and adaptive qualities [Larman, 2004,p. 109]. These abilities are also entitled because of the unpredictability and creativity of an IT system [Larman, 2004,p. 123]. For instance the IT system could encounter altered or additional requirements from the Product Owner continually throughout the development. The main reason why we have chosen to use XP is because of its skilful practices. These practices also make the development adaptive, so it is possible to implement new requirements with quality code even if it is near the end of the development process [Larman, 2004,p. 139].

Scrum combined with XP is appropriate as many Scrum practices are compatible with XP. This includes Scrum meeting, common project room, Sprint planning and a demo in the end of each iteration [Larman, 2004,p. 163] [Mar and Schwaber, 2014].

6.2 Our Use of XP and Scrum

Since we want to use the two development methods XP and Scrum together in this project we will in this section take a look at what the two development methods have in common while also taking a look at the benefits and drawbacks when combining the two methods. The reason we look at these methods in this way is so we can argument for our use of the two methods.

Both development methods are agile and are therefore more flexible and adaptable when compared to traditional methods. Before each workday both XP and Scrum have short meetings to get an understanding of what have been done and what needs to be done for that day by the team. XP and Scrum both work within short iterations (or sprints) where the developing team will have focus on a specific goal. The time span for the two method's iterations are, however, different. XP´s iterations are generally shorter than the sprints typically are in Scrum. When using XP inside a project that has Scrum as framework (like we do) the difference in time span for the iterations can potentially create problems in the programming part of the project.

When using the framework of Scrum it is optimal to only have one product owner attached to the project, however this leads to another conflict when Scrum and XP gets combined. In XP you will have more than one product owner attached to the project to help the

developers understand the problem domain. When combining the two methods it is necessary to secure that the one customer attached to the product will be able to represent all the different users and customers of the project.

The group decided to mix these two methods because Scrum is designed to run project in the typical agile way but does not have anything specific techniques or practices on how to approach programming which is a big part of this project. We felt that we needed practices from somewhere else to follow in the programming part of the project - hence we decided to include XP. To ensure that the quality of the code was as good as possible the team used various practices from XP. For instance we utilized pair programming, continuous integration, refactoring and collective code ownership [Larman, 2004].

6.3 First Sprint

This section will consist of the documentation on how we structured and planned two weeks of our project. These specific two weeks were running from 31/03/2014 to 11/04/2014. The first sprint will primarily concern our pre-coding processes and tasks along with a few of the development tasks and processes. Reading this chapter gives an idea on how project management tools and practices were used, and how the product looks after the first sprint.

The Sprint Goal

Before the sprint started we had to find out what the goal for the sprint should be. We sat in a meeting and discussed in the group what should be our goal; this meant we discussed where we should be with both the report and the system after ending the sprint. The conclusion was the following as our goal:

Log-in function to the system, a simple graphical user interface to start with, a database to the system and a function to manage stock.

The way we came to these goals for our system was we evaluated it based on the interviews we held with our stakeholders/product owners. We are aware that this is not the exact way to plan sprints optimally, however because our stakeholders were not willing to commit too much time, we decided to decide and set the goals as best as we could for our sprints, based on the feedback and information we had gotten from our stakeholders. We then elicited requirements for the system and prioritized those that would be most important for the system. This understanding was reached just before the sprint started, which meant the group had an understanding of what our stakeholders wanted and what the problem domain looked like.

The Sprint Tasks

When the group had an understanding of the requirements from the interviews and the problem domain, we had a discussion of which things was essential to in order to reach the goal of the sprint. In the following some of the tasks we came up with are listed; some of

the tasks are what is called spawned tasks. A spawned task is a task, which comes when the developers stumble upon a problem, which cannot be solved with only the current task, and is so important that it has to be done to complete the wished functionality. Although it is important to remember that the task has to be essential to reach the sprint goal, otherwise it should be saved for later. Here are some of the tasks:

- Benefits and drawbacks in using both SCRUM and XP
- Theory chapter on burn down charts
- Conduct navigation diagram
- Make the ER-diagram of database
- Sketching system architecture and design of the system
- Create a formal list of general code rules
- Create a function that is able to create, delete and edit items
- Create the database to the system
- Theory behind requirements gathering
- Research how to document sprints
- Create a log-in and log-out function to the system
- Create a function to view items in stock

For the entire list of tasks see section 6.3

Task Estimation

The tasks were estimated by using poker planning (see section 2.5.2). The way we did this was to have everyone in the group giving their estimation of how long it would take them to complete a task. Those with respectively the highest and lowest estimation had to argue for their estimated hours. By doing this we sometimes reached an understanding that those in the middle of the estimations did not think of. From this the group would have a discussion and evaluate a satisfactory hour-estimation to complete a task based on the presentation of the two members.

Sprint Backlog

We used the sprint backlog from Scrum to keep track of our tasks and their progress, a picture of the entire sprint backlog from the first sprint can be seen on Figure 6.1. We made the priority list after a ranking system we made. The ranking system goes as follows:

- 1 = Highest priority
- 2 = Higher Priority
- 3 = Medium priority
- 4 = Lower priority
- 5 = Lowest priority
- X = No priority yet
- 10 = Completed

Besides that we had a status system too. It goes as follows with corresponding colours:

- In progress (Yellow)
- Completed (Green)
- Cancelled (Red)
- Feedback (Blue)
- Hold (Purple)

Besides these two factors we had other column categories to keep track of during the sprint. Examples are: the originator of the task, current responsible persons, estimated hours and actual hours spent, and a few more. All this made it possible for us to track who had done what, and which task currently was up for grabs. The complete sprint backlog can be seen at Figure 6.1.

		Priority	Est. hrs	Act. hrs	Category	No. of resp	STATUS	ORIGINATOR	RESPONSIBLE(S)
2	User/admin must be able to edit user	1	12	8	GUI Code	2	In progress	Mikkel, Rasmus	Mikkel, Rasmus
3	User: Rights & Roles	3	16	4	Code	2	Hold	Mikkel	Mikkel
4	Admin must be able to update stock	5	16	15	Code	2	In progress	Jesper, Adam	Jesper, Mikkel, Adam
5	User: Retrieve Forgotten password	5	12	7	Code	2	Completed	Jesper, Adam	Jesper, Adam
6	User must be able to use the functionality on The Stock Frontpage	10	10	11	GUI Code	1	Completed	Thomas	Thomas
7	Bar: Create/delete/edit bar	10	14	0,5	Code	2	Completed	Roald, Thomas	Roald, Thomas
8	Benefits/drawbacks in mix of SCRUM/XP?	10	7	7	Discussion	2	Completed	Roald	Roald
9	Burn down chart - Theory	10	3	5	Theory	1	Completed	Thomas	Thomas
10	Conduct Navigation Diagram	10	20	14	Architechture	1	Completed	Rasmus	Rasmus
11	Count: Create/delete/edit count	10	14	1	Code	2	Completed	Roald, Thomas	Roald, Thomas
12	Creating the menu for the system	10	2	6	GUI Code	2	Completed	Morten, Roald	Morten, Roald
13	ER-diagram	10	35	15	Method	7	Completed	Group	Thomas
14	ER-diagram theory	10	9	13,5	Theory	1	Completed	Morten	Morten
15	Group: Architecture & design	10	49	34	Analysis	7	Completed	Group	Morten
16	Group: Create formal list for general code rules	10	6	3,5	Group	7	Completed	Group	Mikkel
17	Item: Create/delete/edit item	10	18	10	Code	2	Completed	Roald, Thomas	Roald, Thomas
18	Made gui for frontpage and fix navbar	10	8	8	GUI Code	2	Completed	Morten, Roald	Morten, Roald
19	Make Database	10	16	4	Code	1	Completed	Mikkel	Mikkel
20	Requirements gathering (method, DEB og SWEBOK)	10	4	5	Theory/M	2	Completed	Adam, Jesper	Adam, Jesper
21	Research sprint documentation	10	4	9	Project Task	1	Completed	Rasmus	Rasmus
22	Stock: Create/delete/edit stock	10	18	6,5	Code	2	Completed	Roald, Thomas	Roald, Thomas
23	Stock: Receive new items	10	12	3	Code	2	Completed	Thomas	Thomas
24	System Requirements (formal list of requirements) (analysis of interviews)	10	10	25	Analysis	2(3)	Completed	Adam, Jesper	Adam, Jesper, (Roald)
25	User: Create/delete/edit user	10	18	15	Code	2	Completed	Mikkel	Mikkel, Jesper
26	User: Login/logout	10	10	12	Code	2	Completed	Jesper, Adam	Jesper, Adam
27	View tables (spawned task)	10	6	6	Code	2	Completed	Roald, Thomas	Roald, Thomas
28	Burn down chart - Method	X	2	0	Method	1	Cancelled		
29	Documentation of 1st sprint	X	3	0	Project Task	1	Cancelled		
30	Function list (OOA&D)	X	5	0	Report	1			
31	Group: Which statistic to track	X	14	0	Group	7	Cancelled		
32	Make PHP send forgotten password	X	X	0	Code				
33	Research: Counting of stock before/after bar opens	x	2	0	Extern Contact	1			
34	Research: How do they recieve items?	x	2	0	Extern Contact	1			
35	Research: What statistic to track	X	?	0	Research	1			
36	Statistic: Individual items	X	X	0	Code	2			
37	Stock: View other bars' stock	X	X	0	Code	2			
38	**GOAL = Login, Simple GUI, Database, Stock management**								

Figure 6.1. Our sprint backlog of the second sprint

Burn Down Chart

Furthermore we used another Scrum method called sprint backlog graph, or more commonly called burn down chart (see section 2.1). This methodology was used to visually display the progress of our work with daily updates. The burn down chart was an A3 paper we had drawn on and hung on the wall at the back of our group room for everyone to see and follow. On the burn down chart we measured two things: one being our actual progress, meaning how many man-hours there were left before every task would be complete, and the second thing we measured was the ideal progress by estimating how many hours we would have during a normal workday. The ideal progress shows how the work process would look if we did not encounter any risks or problems. The chart gave the group members a quick view of the current progress, and it was pleasing to see the graph coming closer to zero, which boosted the morale of the group. The burn down chart can be seen at Figure 6.2.

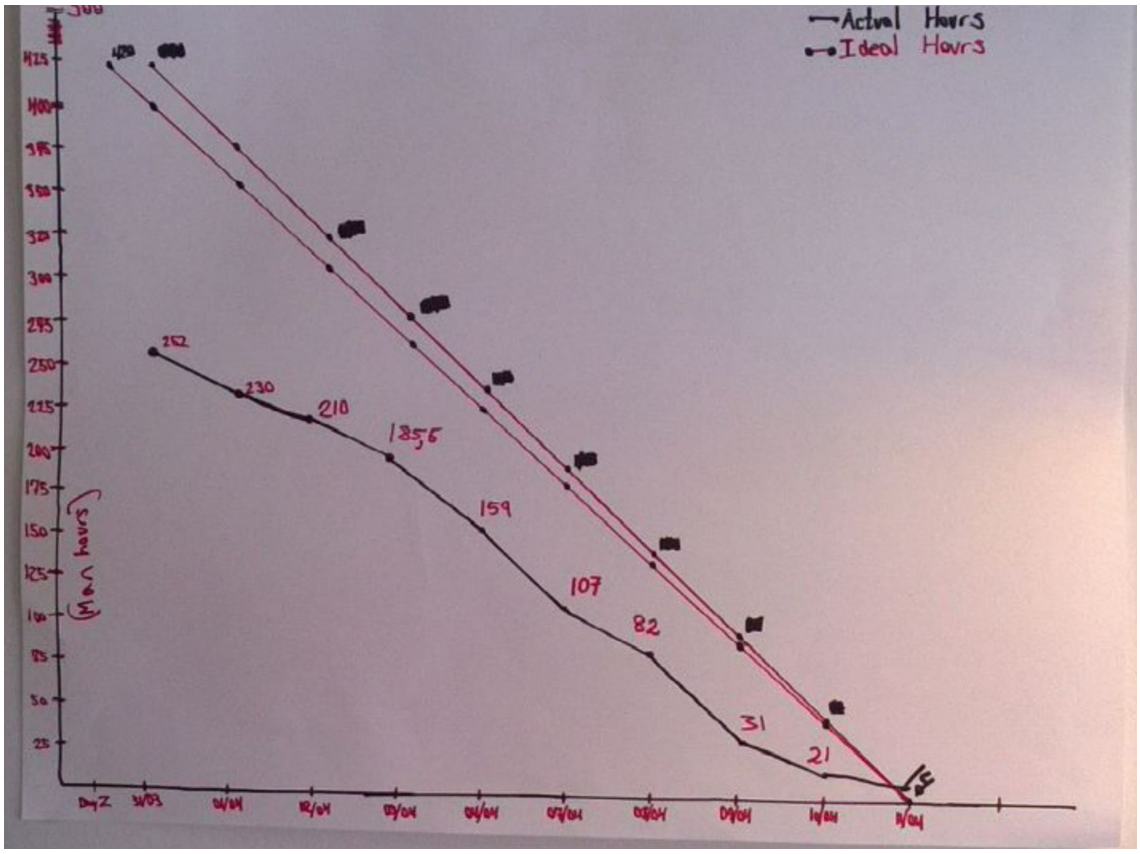

Figure 6.2. Our burn down chart from the first sprint

Completed Features

This subsection will summarize some of the system features we completed out of the given tasks listed earlier.

Top Navigation Bar At Figure 6.3 is a cropped picture of how our top navigation bar looked when we had it completed for the system. Based on the requirements we had, we placed the different options on the navigation bar accordingly to how they work at the bars, and what information they need when they are working.

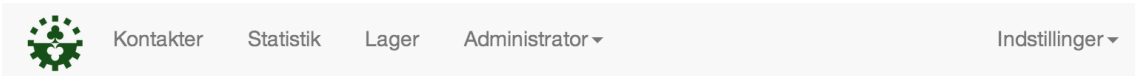

Figure 6.3. The top menu bar in the system

Edit User At Figure 6.4 is the page for editing a user. This function was made for the user and admin, because they should be able to edit their profiles in the system, in case the current information needs to be changed.

Navigation Diagram n this sprint we did our first version of our navigation diagram, however rework on the diagram have been done in the next sprint for better adjustment. The navigation diagram shows how the pages link together throughout the entire system,

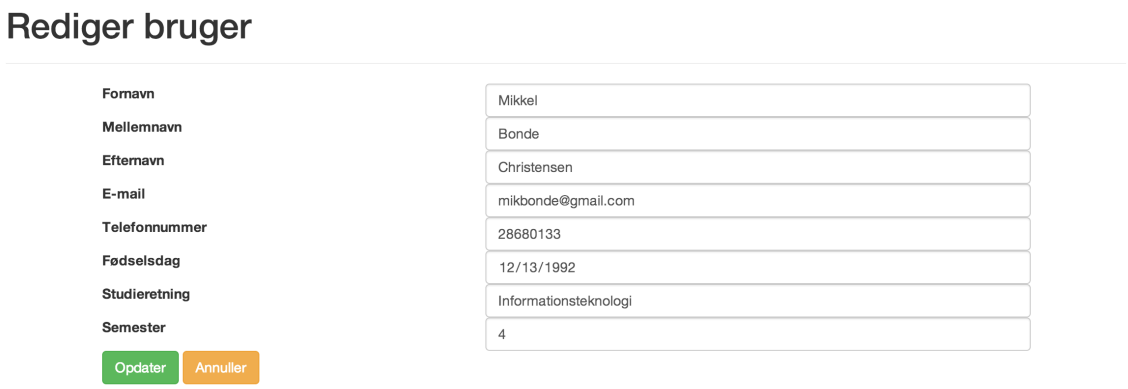

Figure 6.4. The page where you can change the personal information about your account

and how the system structure was made into an easy understandable system-hierarchy. The navigation diagram can be found under the section design, see section 4.2.

Login We finalized the entire login front page. We still had an issue with the different right and roles of the users on the different pages in the system at this time being, but the login functionality was complete.

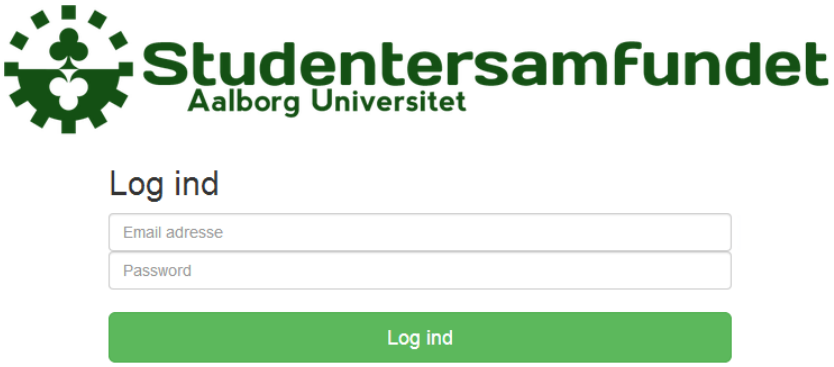

Figure 6.5. Our login section

Stock We got some functionalities down, such as viewing data from the database, being able to create/delete a stock. But this was still very much work in progress. For instance the GUI still needed a brush-over, therefore the work on the stock page was continued in the next sprint. Sadly we do not have a picture of the work in this version, but a later version of Stock page can be found in the implementation section, see section 5.5.

ER-Diagram We got our first version of our ER-diagram out, which is used to map the structure of the database. This meant we could easily execute the database, but as it turned out the database design needed rework, which meant we did rework on the ER-diagram in the next later sprint. To see the ER-diagram go to 4.4.

Database We conducted the entire database code and got the entire Relationship Database Management System up and running. For more on this see under the section implementation, see section 5.5

The entire list of system features is described in the implementation, see section 5.2 for a complete walkthrough of the system.

6.3.1 Reflection of First Sprint

This subsection will reflect on how the entire sprint went as a work process, and what we thought the methods and practices ourselves.

After the sprint we could see how our estimation had went in the sprint backlog. In the beginning of the sprint we had planned that it would take 252 hours to complete all the tasks listed at that time. Throughout the sprint more and more spawned tasks turned up, meaning that the total time grew. We had estimated some spare hours in the beginning, but those hours were not enough and we ended up being under pressure and we therefore still had a few tasks to complete at the end of the sprint. At the end of the sprint we could see that we had a total of 377 hours in estimated and 248 hours in actual hours, which indicates that we had estimated too much time for the tasks. We decided that we wanted to take this newfound knowledge into account while doing poker planning for the next sprint.

During the sprint we also encountered tasks we cancelled during the stand-up meetings. We either found them irrelevant at the current point, or at the end of the entire sprint we marked some tasks as priority X, meaning it could not be a priority at that time because of the time pressure. Those marked as X were tasks we needed to add as further work in another sprint. Some of the discontinued/further work tasks were:

- Make the function list for the system
- Find out which statistic we want the system to track
- Make the system send users forgotten password
- Find out how the product owner receive new items
- Make a function that can view other bar stocks in the system

Sprint Retrospective

After the sprint we held a sprint retrospective meeting to identify what went well, what went bad and what we could have done better. This will be a list of what we agreed on. The entire retrospective meeting was a long debate, which we spent almost a day on deciding what practices to continue, discontinue and improve on.

What Should We START Doing?

- We want to improve task estimation, and we want to do it because the current tasks are too complex. If possible we will split the tasks into smaller tasks we can more easily estimate them.

- Create better task descriptions. Some task descriptions say too little about what should be done within the task. Sometimes this was a problem because the tasks were too big and complex.
- When we think of a spawned task/or work on it, we should to write it down immediately. We had problems with the spawned tasks being forgotten because we did not write them down as they spawned. This means some of the hours we spent was not recorded in the sprint backlog, or was just noted on a different yet similar task.
- More communication while coding. It was still unclear which group was working on what functions and in which document. We wanted to solve this by simply getting better at asking each other and getting communication to flow better across the table. This is done to prevent groups overwriting the same files.
- Make a coding plan, instead of just jumping into coding without one.

What Should We STOP Doing, and Why?

- The pair programming practice from XP. This practice had multiple flaws in our work process; some in the group did not like the fact they had to change position every hour, while others did not like that they had to do everything on one computer. We considered creating our own practice, which worked more as work-partners rather than pair programming. Meaning we would still have two who worked on a solution for the same task but on different computers. The partners would work more as guidance to each other, i.e. they would share comments and knowledge on each other's work.

What Should We CONTINUE Doing, and Why?

- We want to continue using the burn down chart practice from Scrum because it gives a nice overview of the group's current work progress.
- Estimation of tasks. We want to continue the poker planning process, even though we found it rather hard to give a real estimate. But hopefully we will be more accurate for the next sprint, with our new hands-on experience.
- We created a Lo-Fi sketch of the design just before a person started coding a specific part of the system, so that the developing members knew exactly what was relevant to code after. This also worked as a sort of task description of the functionalities.

This section described how we started the sprint, defined our goal, decided on tasks, how we handled spawned tasks and time pressure, which features was conducted, and finally how the group reviewed the entire process in the retrospective and came up with constructive feedback to use and work upon for further and better progress with the project.

6.4 Second sprint

This section will be documentation on how we structured and planned two weeks of our project. These specific two weeks were running from 22/04/2014 to 02/05/2014. The Second sprint concerns itself with the development tasks that was completed in order to create the final system.

After reading this chapter the reader will know which project management tools and practices were utilized, and how the product was looking after the second sprint.

The Sprint Goal

Just like the first sprint we sat down to discuss what should be the goal for our second sprint. And what the product should look like after this sprint. We came to the conclusion that we wanted our system fully running and should be able to do all the requests from the previous sprint, plus a statistics functionality.

We defined our second sprint goal to be: A fully running stock system with statistics.

We decided on this goal mainly because of the status of the system at the time the meeting was held. At that point we were also considering other functionalities such as a calendar function, however based on our prioritirization of our requirements we decided to not create this functionality.

The Sprint Tasks

Since we got an understanding of our product owner's requirements in the first sprint, we focused on what was currently needed to reach this sprint's goal and nothing else. Only at the end of the second sprint we decided to create some report related tasks, since the current code tasks were too few for the seven man workforce. Here are some of the tasks from the second sprint; the entire list can be seen under section 6.4.1.

- Rework ER-diagram
- Design statistics page (Sketch)
- Admin must be able to update stock
- User must be able to keep track of items on statistics page
- Implement user rights for all system pages
- Finish the Edit User page
- Integrate the whole system together
- Refactoring of existing code according to bootstrap
- Documentation of existing code
- Technologies walkthrough
- Create common thread throughout the report

Task Estimation

We did poker planning just as described in the first sprint. The difference this time was clearly noticeable. The group members was in general not as far from each other in the hour estimation this time. This meant that the entire group has gotten more experienced at estimation, and this was definitely showing on how we estimated the tasks for this sprint.

6.4.1 Sprint Backlog

The sprint backlog practice was used like we did in the first sprint; that means we used the same procedure and priority system. This was simply because we felt the system worked well for us. The only downside was that our group members were sometimes slow or forgot to add spawned tasks in the sprint backlog in the first sprint. This downside was improved in this second sprint, and we enjoyed using the sprint backlog as a status tool for our progress. See the second sprint backlog in figure 6.6.

	Tasks	Priority	Est. hrs	Act. hrs	Category	No. of resp	STATUS	ORIGINATOR	RESPONSIBLE(S)
2	Describe the design method	1	20	20	Method	1	In Progress	Roald	Roald
3	Writing technologies walkthrough	1	14	3	Report	1	In Progress	Morten	Morten
4	Burn down chart method from 2nd Sprint	2	6		Method	1			
5	Sprint 2nd documentation	2	22		Method	1			
6	Navigation diagram theory&method. Expand to the new functionalities.	10	10	4	Method	1	Completed	Adam/Jesper/Roa	Adam/Jesper/Roald
7	Admin must be able to update stock	10	16	22	Code	1	Completed	Jesper/Adam	Mikkel
8	User must be able to keep track of items' statistic over time.	10	24	40	Code	2	Completed	Thomas	Thomas
9	Refactoring of existing code according to Bootstrap	10	15	8	Code/GUI	2	Completed	Roald/Adam	Mikkel
10	Integrate the whole system together	10	10	3	Code	1	Completed	Mikkel	Mikkel
11	Implement user rights in whole system	10	20	2	Code	2	Completed	Thomas	Mikkel
12	Place code files in relevant folders	10	6	2	Code	1	Completed	Mikkel	Mikkel
13	Write the php part of edituser.php	10	6	12	Code	2	Completed	Morten	Morten
14	Rework ERD	10	2	1	Method	2	Completed	Adam/Jesper	Adam/Jesper
15	Common thread between our report's current content	10	14	9	Report	1	Completed	Adam	Adam
16	Documentation of "existing" code	10	15	3,5	Code	1	Completed	Jesper	Jesper
17	Design the statistics-page	10	5	4,5	Design	1	Completed	Rasmus	Rasmus
18	Make the register-page announce "user registered"	10	10	2	Code	2	Completed	Thomas	Thomas
19	Make the register.php-page save roles	10	5	1	Code	1	Completed	Mikkel	Mikkel
20	Design stock frontpage	10	10	2	Code	1	Completed	Morten	Morten
21	Adjusted css to toggle a fixed menubar	10	1	0,5	Gui-code	1	Completed	Morten	Morten
22	Optimization of Report structure and organizing LaTeX	10	5	7	Report	2	Completed	Jesper/Adam	Jesper/Adam
23	Make PHP send mail (forgotten password)	X	30		Code	2	Cancelled	Jesper/Adam	Mikkel
24	Find material about object-oriented programming	X	2	0	Research	1	Cancelled		
25	Group: Research object-oriented programming	X	5	0	Research	7	Cancelled		
26	Rewrite code to objectoriented	X	20	0	Code	2	Cancelled		
27				146,5					
28	**Goal = We want a fully running stock system with statistics.**								
29	**WHEN CODE: GUI, Code, Document, Done**								
30	**WHEN TASK: REFLECT**								

Figure 6.6. Our sprint backlog of the second sprint

6.4.2 Burn Down Chart

In this sprint we also used a burn down chart, and it worked just as well for us this time like it did in the first sprint. Just as the sprint backlog we liked this practice because it gave us a good visual status of our current progress. The burn down chart practice procedure was the same as in the first sprint. The second sprint burn down chart can be seen on figure 6.7.

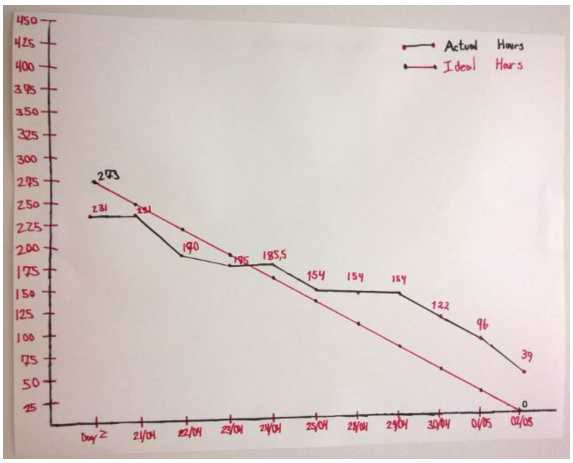

Figure 6.7. Burn down chart of our second sprint

6.4.3 Completed features

This subsection will summarize some of the system features we completed out of the given tasks listed earlier. Further, this also covers some of the documentation and report relevant tasks we did in this second sprint.

Rework ER-diagram In this sprint we finalized the database design, and therefore had to rework the ER-diagram to map our new design features. We finalized our ER-diagram in our second sprint. To see our final ER-diagram see figure 4.4.

Design Statistics Page (Sketch) In this sprint we continued designing sketches before coding. As such we created some Lo-Fi sketches that the coding team could base their work on. On figure 6.8 the Lo-Fi prototype of the GUI for our system's statistics page can be seen.

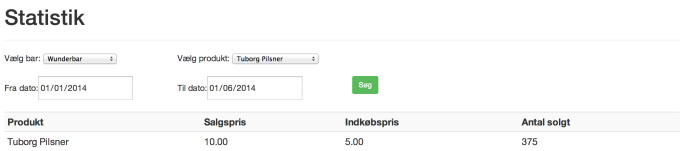

Figure 6.8. A sketch of the statistics page

Admin Must Be Able to Update Stock This was one of the heaviest and most difficult code tasks we have had during this project. Essentially this task should have been broken down into smaller tasks since it was way too complex. This task was the task we did not finish in the first sprint. The most difficult to code was to get the stock update page to work properly. See figure 6.9 for a picture of the update stock function.

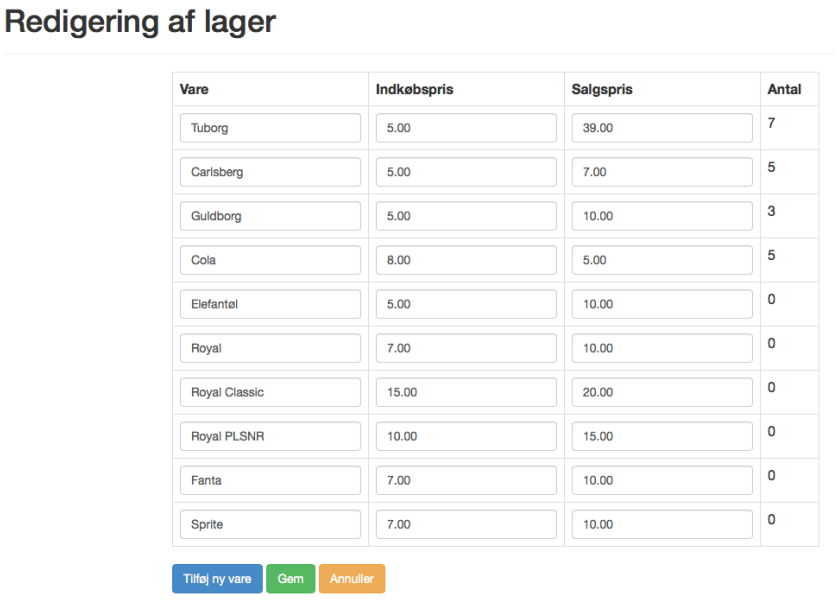

Figure 6.9. Picture of the update stock function within the system

User Must Be Able to Keep Track of Items on Statistics Page This was the new feature we wanted our system to include at the end of this sprint. The code task for this was difficult to complete since it has been rather complex. The final result was a statistics page with that would enable the user to search through former sales statistics. See 6.10 for a picture of the statistics page.

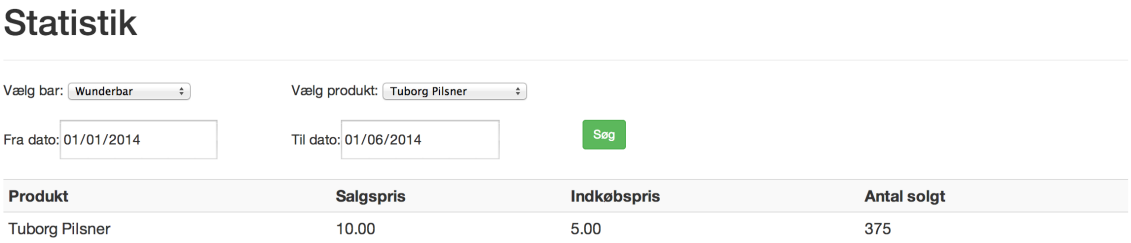

Figure 6.10. Picture of the statistics page of the system

Integrate the Whole System Together This task did not take as long as we had estimated it to take (as can be seen on figure 6.6). During this task we made our system accessible through a website. We wanted our system to be hosted on a server for the purpose of giving others access to interact with the system.

Documentation of Existing Code This task was a spawned task and we spawned it after we had run out of coding tasks for everyone to have. We knew that it was something we were supposed to do at some point, so it was fitting that we just spawned this task when everybody did not have something new to work on. The documentation followed a standard called PHPDoc [Beaver, 2014].

Create Common Thread Throughout the Report This was another spawned task we made. Its purpose was to create a common thread throughout what was the current state of the report during the second sprint.

6.4.4 Reflection of Second Sprint

This section will reflect on how the entire sprint went as a work process, and what we thought of the methods and practices ourselves.

In the second sprint we calculated that we had 273 hours available during the two weeks, because we would also be busy with two workshops and other course related stuff. We had a tight schedule meaning we estimated our total workload at the beginning to be 230 hours. This left us having very little time for spawned tasks. At the end of the sprint we had estimated work for 236 hours, and this is without the tasks we had to cancel (see figure 6.6 where the cancelled tasks - those marked in red - can be seen). For actual hours we ended up using 147 hours. This meant we had a few tasks left that were still in progress; this left us being 39 hours behind schedule at the end of the sprint.

During the sprint we had to cancel some tasks because we could not find time for them. It was lower priority tasks which could have been nice to have features. Those tasks are marked as priority X in the sprint backlog. Underneath the cancelled tasks are listed:

- Make PHP send mail (forgotten password), we estimated this task to be 30 hours mainly because we knew very little of how we would make the PHP code retrieve the password from the database and mail it to the user.
- Research object-oriented programming. We were thinking about rewriting some of the code using the object-oriented paradigm, since it would supposedly make some functionalities easier to make. However, we decided to keep our procedurally made code because we did not have any more time to spare.
- Rewrite code to object-oriented - same argument as the one above.

Sprint Retrospective

After the second sprint we held a sprint retrospective meeting to identify what went well, what went bad and what we could have done better. This is a list of what we found. The entire retrospective meeting was a long debate, which we spent almost a whole day on deciding what to improve, continue and discontinue.

What Should We START Doing?

- We should describe how much time there has been spent on a task each day; this should be in a column of its own in the sprint backlog. This would be better than only noting down how much time there is left on the current task. By doing this we can keep track on how many hours there have been spent on work each day. This means we could document how many hours have been spent on a spawned task, because one of the problems was we could not see if a spawned task was completed on the same day it spawned; it would not show on the burn down chart.
- We need to be able to make better task descriptions, as some tasks still caused confusion in the second sprint. This lead to the same problem as in the first sprint where some group members were unsure of what they had to do to complete the given task.
- We need to continuously test code to ensure a certain quality of code better.

What Should We STOP Doing, and Why?

- We should stop using pair programming/our partner system because it does not have the wanted effect as it should have. The group ended up not following the practice because it did not prove as efficient due to the fact the group's members are not as skilled programmers as the practice would require.

What should we CONTINUE doing, and why?

- We want to continue with some group members working on the report while others finish the rest of the lacking code.
- We should continue estimating tasks with poker planning because we feel that it engages the group and creates healthy discussions.

- The burn down chart tool is something we want to continue using. The mental effect of having progress visualized seems to have a positive effect on the group.
- We want to continue co-ownership of the code, because we can rewrite every line of code without anyone taking it personal if some of their conducted code gets edited or deleted.
- Refactoring our code is a practice we will continue to use as it enables us to create cleaner code.
- We want to continue using version control. We have grown very fond of the rollback and commit functionalities in Github. Yet we have experienced that version control works better for code than report related material. So in the future we might want to use Github for only the code.
- Code documentation is something we will continue doing since it improves readability in the code.
- We want to continue using continuous integration to ensure that the code is always working.
- By using coding standards we have saved a lot of time on refactoring because each individual developer followed pre-determined standards of coding.

This section described how we started the second sprint, defined our goal, decided on tasks, which features was conducted and finally how the group reviewed the entire process and came up with constructive feedback to use and improve further work processes.

Conclusion 7

In this chapter we will discuss problems we encountered within the project and discuss what could have been done differently. We will then talk about what we would have done in the future, if we have had more time to work on the project. Lastly we are going to conclude upon the project and the problem statement made for the project as well.

7.1 Discussion

In this section we are going to discuss what we felt went wrong in the project and what we think we could have done in a different way.

One of the practices we used in the group was Pair Programming. We tried giving it a chance, however the group found that it just did not seem very efficient, because as the group was programming and came across problems, it would simply take too long time to solve the problem when we had two programmers sitting on one computer. We learned later from meeting with our counsellor that we could solve this problem by programming in Google Docs so both would be sitting at their own computer and could do their research. One of the downsides to programming within Google Docs would however be that there would be no syntax checker to prevent us from making mistakes or typing errors in the code. Had we utilized this method, the Pair Programming method might have been a good practice for the group, however we had finished programming by the time we learned how we might be able to solve the problems we felt we have had with Pair Programming.

In the start of the project we did not know much on how to estimate tasks that needed to be completed within the sprints. We started estimating without having any discussions on how much time tasks would take, but after a sprint retrospective we decided to improve our estimations and thus we started utilizing Poker Planning. We feel that planning and estimations got much better after this. By using Poker Planning we got more constructive discussions about the tasks and what needed to be done, so the outline of the tasks became much clearer as well. If we had used Poker Planning at the start of the project the group might have avoided some problems in the beginning, where tasks were too vague and as such the scope of them was heavily misrepresented in the estimation.

In the group we used Git as version control to make sure that the group could work independently and still have an easy way of sharing all files. Utilizing this tool has caused multiple lost man-hours due to errors that would occur and had to be fixed before some could continue with their work. The good thing about using it, was it was always possible to perform a rollback within the system in case someone had done a mistake when they

committed their changes/files to the master branch, the problem was that sometimes even this would give us errors that we had to look into to fix.

We also utilized collective code ownership within this project, which enabled the group's members, access to change every other member's code. While this practice made it easier for us to quickly perform a change in someone else's code, it also meant we came across problems. We lost many man-hours trying to fix code that had been changed by another team member, and it was not even enough to look in Git and rollback the changes. While we did come across this problem that halted our development for quite some hours, this practice was still efficient because we could quickly look into each other's work.

After we had implemented Poker Planning for estimation we also started using a burn down chart. The burn down chart became a great help for the group, as looking at it hanging on the wall had a positive effect on the work moral of the group. Being able to visually see how far you are according to the plan was great to keep a much better overview of how far we were in completing our tasks set for the iterations.

We encountered a problem when sketching our Lo-Fi prototype of the system, because it ended up being a problem having all the sketches ready before we started programming. This resulted in some waiting time, because we had to wait with programming the Graphical User Interface before the sketching was done.

Lastly we did not perform any sprint reviews with our stakeholders due to lack of time. Had we done this, we might have had other requirements, which could have had an influence on the way we designed the system. Using sprint review is also something we could have used to ensure the system is fully according to how the stakeholders envisioned the system, and based off their feedback we would have adapted the system accordingly.

7.2 Future work

This section is about putting our project into perspective on how our system could be optimized, had we more time to spent on it.

First of all it is relevant to clarify how the system could be optimized regarding security. In our system we have intentionally left out security of both our PHP code and our database. For instance the users' exact password is inserted into the database, which means that we are able to explicitly see which passwords they use. It is also appropriate to prevent our database from SQL injections, because we know it would also be a problem if we have no security here as well.

Another activity in the future work would be to conduct usability tests with some representative users to evaluate the system. We would then be able to fix the discovered usability problems within the system.

If we had more time, new features could be added to make the system more generic. Here it will be noteworthy to mention a feature, which will make it possible for the user to add or edit columns in the bars stock, so that the user could customize it to their own specific

needs. We had also thought about having a function that would tell the user how much they needed to order, before filling the bars stock to a minimum they have set for the stock. In that way they would always know how much they at least needed to order every time to fill up the stock.

7.3 Conclusion

This section will summarize the group conclusions and experience, which we made throughout the entire project.

In this project using a combination of the Scrum and Extreme Programming as software engineering approaches has been used to develop an IT system. By using Scrum and Extreme Programming, a better overview of the scope of the project has been achieved, thus allowing us to structure the work within the project. Experience with the use of software engineering approaches has contributed to a better consciousness of both which approaches to choose, and moreover what techniques and practices from the software approaches to use in the specific context to get the most out of the approach, that has been taken in this project.

Learning the basics of databases, including design of ER Diagrams, normalization and joining tables has contributed to partly a more efficient database and partly a database, which is more comprehensible for other developers. Furthermore continually refactoring the code and performing unit tests ensured the system's quality. These methods concluded a database system that facilitates a stock management within the bars of Aalborg University.

Bibliography

Beaver, G. (2014, May). *phpDocumentor Tutorial*. http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_-phpDocumentor.howto.pkg.html: PHPDoc.

Beighley, L. (2007). *Head First SQL*. O'Reilly.

Benyon, D. (2010). *Designing Interactive Systems* (2nd ed.). Pearson.

Berg, T. and R. Pooley (2013). Rich pictures: Collaborative communication through icons. *Systemic Practice and Action Research 26*(4), 361–376.

Boehm, B. and R. Turner (2003). Observations on balancing discipline and agility. In *Agile Development Conference, 2003. ADC 2003. Proceedings of the*, pp. 32–39. IEEE.

Bootstrap (2014, May). *Bootstrap*. http://getbootstrap.com/css/#less: Bootstrap.

Bootstrap (May, 2014). *CSS*. http://getbootstrap.com/css/: Bootstrap.

Glogler, B. (2011, March). *Scrum Checklist 2011. bor! sgloger Wien*.

Harjono, J., G. Ng, D. Kong, and J. Lo (2010). Building smarter web applications with html5. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, pp. 402–403. IBM Corp.

Jomfru Ane Gade Portalen (2014, May). *Ud at feste*. http://jomfruanegade.dk/steder/UdPortalen.

Koskela, J. (2003). Software configuration management in agile methods. *VTT PUBLICATIONS*.

Kvale, S. (1996). *Interviews: An introduction to QUalitative Research Interviewing*. SAGE.

Larman, C. (2004). *Agile and iterative development: a manager's guide*. Addison-Wesley Professional.

Mar, K. and K. Schwaber (2014, March). *Scrum with XP*. http://faculty.salisbury.edu/ xswang/Research/Papers/SERelated/scrum/ScrumXP.pdf.

Mathiassen, L., Munk-Madsen, Andreas, P. Axel Nielsen, and J. Stage (2001). *Objektorienteret analyse og design* (3rd ed.). Marko.

Mittal, N. (may, 2014). *The Burn-Down Chart: An Effective Planning and Tracking Tool*. http://www.scrumalliance.org/community/articles/2013/august/burn-down-chart-Scrum
Alliance.

Nixon, R. (2012). *PHP, MySQL, JavaScript & CSS*. O'Reilly.

Persson, J. S. (2014). *Lecture 11: Project Management, scheduling and estimation*.

Persson, J. S. (February, 2014). *UNIFIED PROCESS & AGILE OR TRADITIONAL?* https://www.moodle.aau.dk/pluginfile.php/250883/mod_resource/content/1/SOE_-05-UP__agile_v_traditional.pdf:
AAU.

Pressman, R. S. (1997). *Software Engineering: a Practitioner's approach*. McGraw-Hill Companies.

Schuh, P. (2007, March). *Agile configuration management for large organizations*. http://www.ibm.com/developerworks/rational/library/mar07/schuh/index.html.

Schwaber, K. and J. Sutherland (2012). The scrum guide–the definitive guide to scrum: The rules of the game (2011). *URL http://www. scrum. org/storage/scrumguides/Scrum% 20Guide*.

Techterms.com (2014, May). *Techterms: Session*. http://www.techterms.com/definition/session: Techterms.com.

Teorey, T. J., S. Buxton, L. Fryman, R. H. Güting, T. Halpin, J. L. Harrington, W. H. Inmon, S. S. Lightstone, J. Melton, T. Morgan, et al. (2008). *Database Design: Know It All: Know It All*. Morgan Kaufmann.

The PHP Group (2014, May). *Language Reference*. http://www.php.net/manual/en/langref.php: The PHP Group.

Twitter (2014, April). *Bootstrap*. www.getbootstrap.com.

Van Der Hoek, A., R. S. Hall, D. Heimbigner, and A. L. Wolf (1997). *Software release management*, Volume 22. Springer-Verlag New York, Inc.

W3-Schools. Sql injection.

Digital Copy of Project A

System Prototype Design B

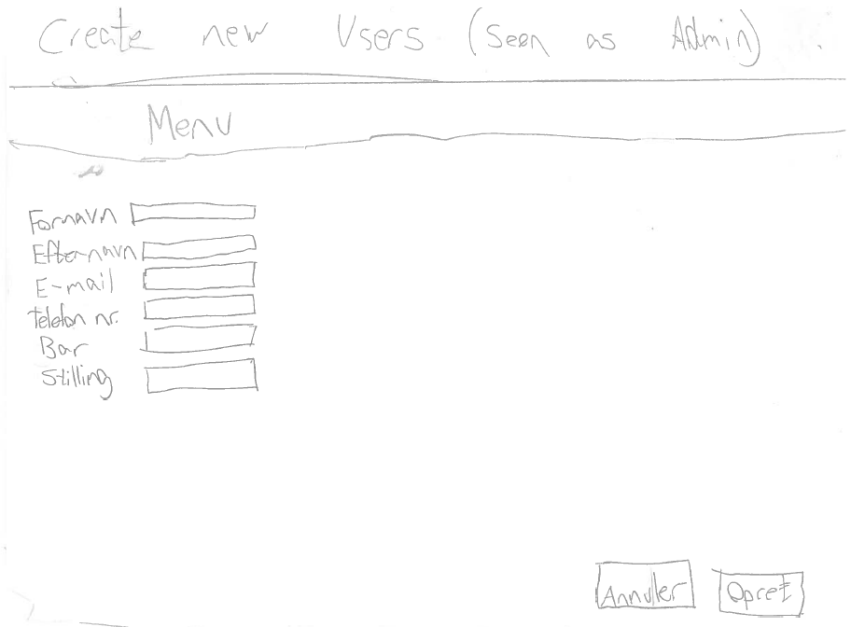

Figure B.1. Lo-fi prototype of the create user page

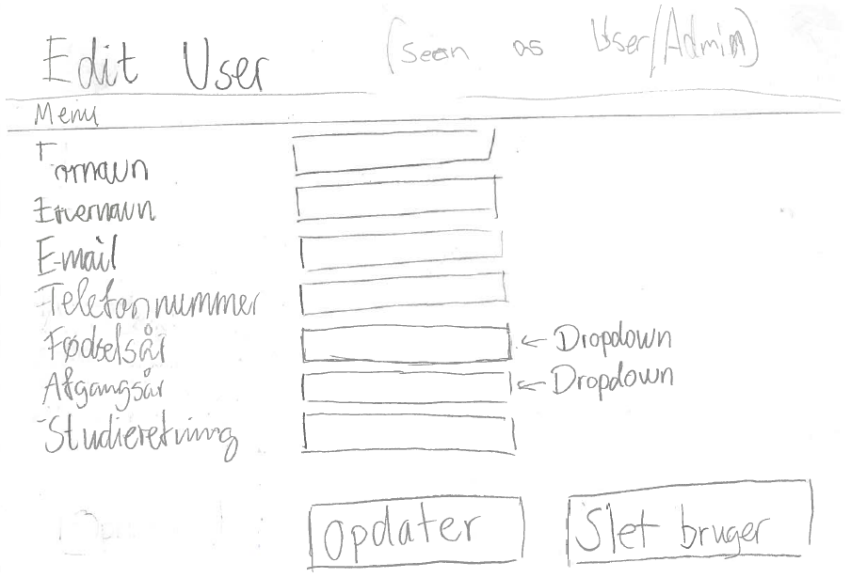

Figure B.2. Lo-fi prototype of the edit user page

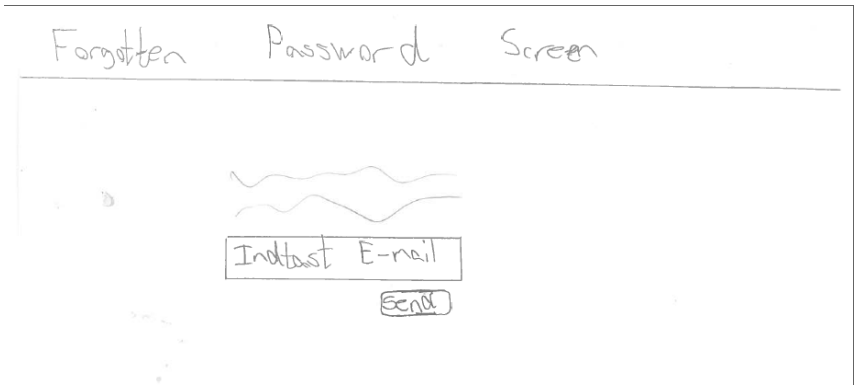

Figure B.3. Lo-fi prototype of the forgotten password page

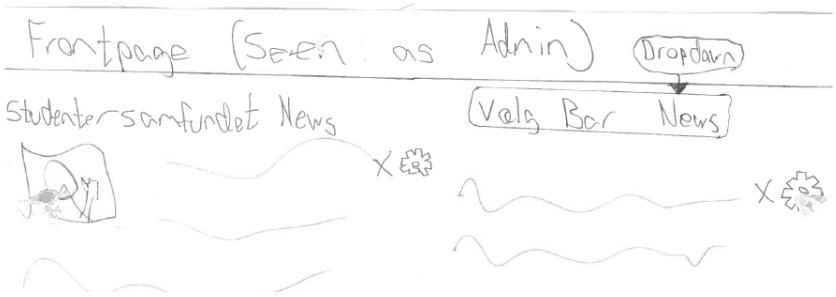

Figure B.4. Lo-fi prototype of the front page seen as admin

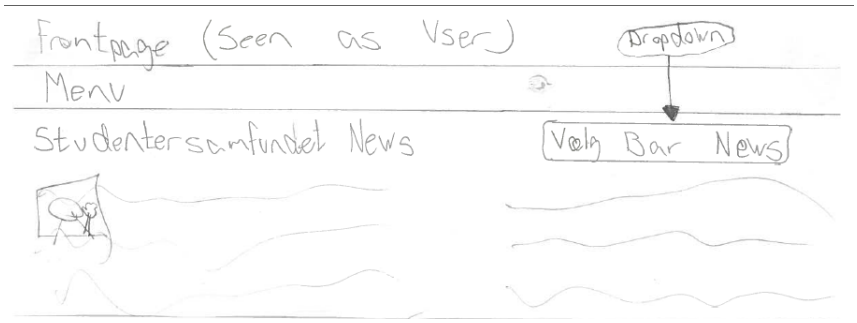

Figure B.5. Lo-fi prototype of the front page seen as user

Figure B.6. Lo-fi prototype of the login page

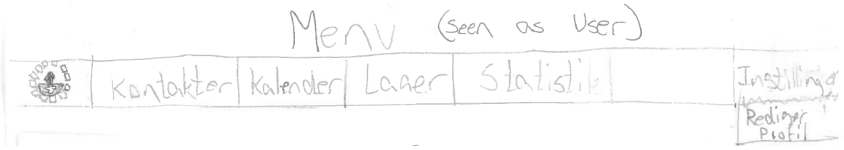

Figure B.7. Lo-fi prototype of the menu seen as user

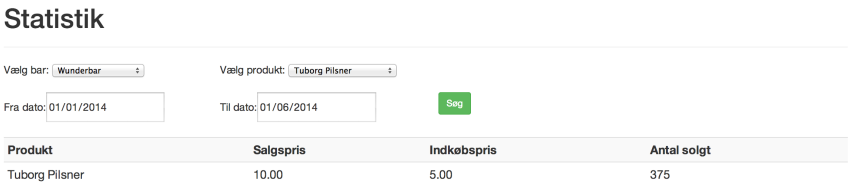

Figure B.8. Lo-fi prototype of the statistic page

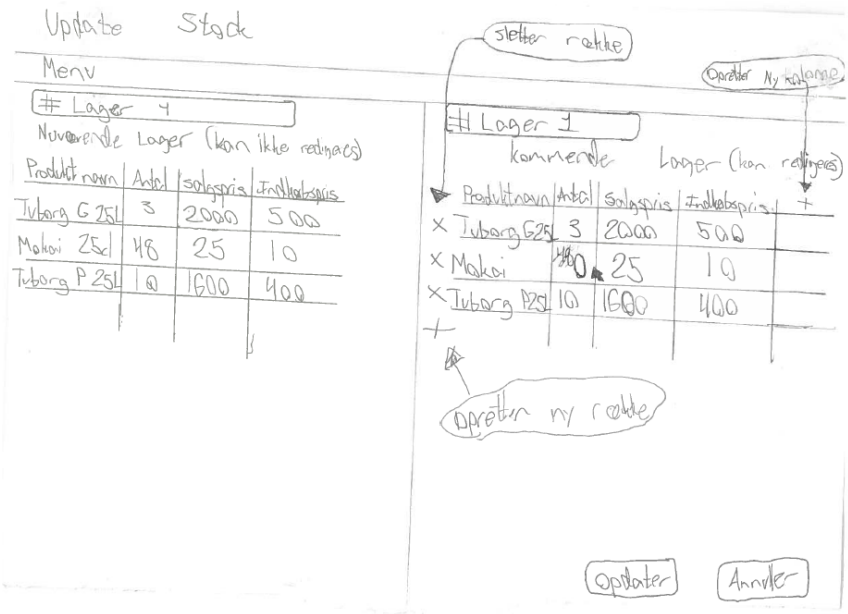
Update Stock
Menu
Lager 4
Nuværende Lager (kan ikke redigeres)
Produktnavn | Antal | salgspris | Indkøbspris
Tuborg G 25l | 3 | 2000 | 500
Makai 25l | 48 | 25 | 10
Tuborg P 25l | 10 | 1600 | 400
sletter række
Opretter ny kolonne
Lager 1
kommende Lager (kan redigeres)
Produktnavn | Antal | Salgspris | Indkøbspris | +
Tuborg G25l | 3 | 2000 | 500
Makai | 48 | 25 | 10
Tuborg P25l | 10 | 1600 | 400
opretter ny række
opdater
Annuller

Figure B.9. Lo-fi prototype of the update stock page

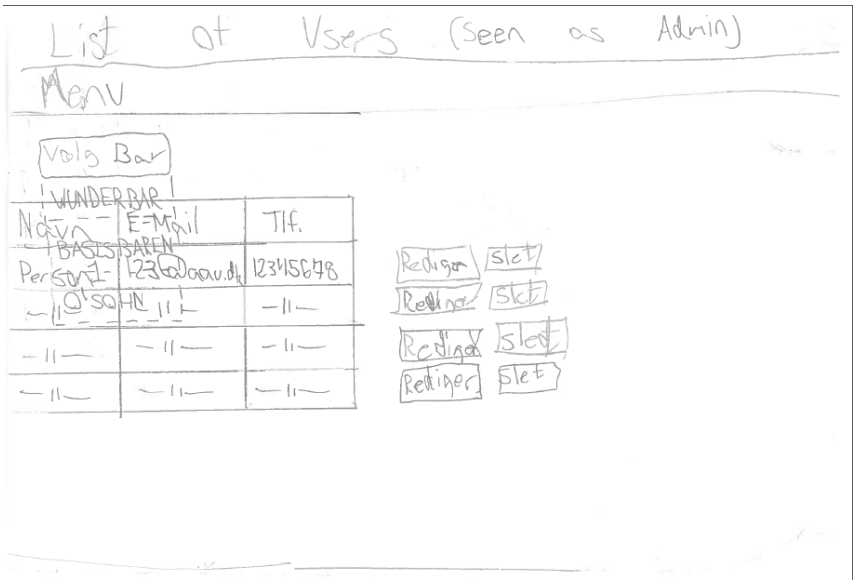

Figure B.10. Lo-fi prototype of the user list seen as admin

Feature List for the System C

User Management

- Login/logout
    - **Goal:** you should be able to log in to the system by using a user name and password; this is to keep unauthorized people out.
    - **Function:**
        - Login
            - It is need for a user to log in for that user to use the functions of the system.
        - Logout
            - A user needs to be able to log out when they are done using the functions of the systems.
- Create/delete/edit user
    - **Goal:** Be able to create, delete and edit user.
    - **Function**
        - Create user
            - You will need a user to use the system; only administrators can ad new users.
        - Delete user
            - An existing user needs to be able to be removed from the database when that user is no longer a part of a bar. This can only be done by normal users
        - Edit user
            - A user needs to have the ability to edit they own user.
- Rights & Roles
    - **Goal:** Create and assign role(s) to a specific user.
    - **Note**: It needs to be possible to have a role in more than one bar and these roles not to be the same
    - **Function:** User roles/groups
      Create/delete/edit roles
            - A role needs to be creating in the system and edit or delete in the same manger. This is a right only admins have.
        - Assign roles to users
            - Admin gives a user a role in a bar or edit the role of that use.
- Retrieve Forgotten password
    - **Goal**: Get a new password from the system.
    - **Note:** The system needs to make a random new password and send this to the user.
    - **Function:**
        - Change the password to a set of random characters.
            - This code is saved in the database.
        - Send the new password to the user by email.

Stock Management

- Create/delete/edit stock
    - **Goal:** Being able to handle several stocks at a Bar.

        - **Note**: Only admins from the bar can update the bar that they are connected to. This also covers create new stock and delete stock.
        - **Functions**
            - Create a stock
                - You have to be able to create more storage in the system, as this may be necessary for the bars.
                - Change inventory data.
                - You have to change the information of the stocks in the database if you need it.
            - Delete stock
                - If the no longer is a need for the stock in the bar it can be deleted
- Add/remove item to stock
    - **Goal:** remove and add items to the stock
    - **Note:** The way we manage what is in store now, is just the latest count - so that should technically make a "count" which is the latest.
    - **Functions:**
        - When a time is sold or other wise removed from the stock, it should be noted in the system.
        - Add new item to the stock
            - New items shall be added to the systems database.

Item managements

- Create/delete/edit item
    - **Goal:** the option to add, delete and edit items.
    - **Note**: name, description, price and purchase price
    - **Functions:**
        - Create new items
            - Add new items that are not in the database.
        - Delete items
            - Deleting an item if it is no longer being sold in the bar.
        - Edit items
            - It needs to be an option to change information on an item if this changes.
- Receive new items
    - **Goal: H**elps the bars order new items.
    - **Functions:**
        - Send order to the purchaser
            - No direct purchase system, just a shopping list
    - Counting of the stock
        - The system needs to show what is in stock before and after an opening day.
- View other bars' stock
    - **Goal:** gives the user the option to see what other bars have in stock.
    - **Functions:**

        - Stock managements
            - See other bars stocks
            - See inventories for specific items

Menu

- **Goal:** An overview over the contents of the system.
- **Functions:**
    - Contains links to Users, calendar, en stock overview and statistics.
    - Must contain the option to change settings.
        - Admin: create new user.
        - Edit profile.
        - Log out.

Frontpage

- **Goal:** Create overview over news from the Studentersamfundet and a chosen bar.
- **Note:** the user needs to be able to chose from what bar he or she want the bar news.
- **Functions:**
    - Admin: settings for news can be changed by the admins.

Extra

- User
    - Picture of the users
    - Verification of user information - e.g. through student-mail.
- Stock
    - Statistics
        - Individual items statistics
        - Graphs
- Employee Management
    - Overview over employees
        - Name, phone number and Bar
    - Schedule Friday bars.
        - Work Plan
- Communication
    - Wanted product-feature
        - Find the bars that hold the item that the users bars need.
    - Contact x bar
        - Use function to create contact to another bar, then the system automatically checks.
- File Management
    - Upload file